#134 DECEMBER 1987

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

DR. DOBB'S JOURNAL

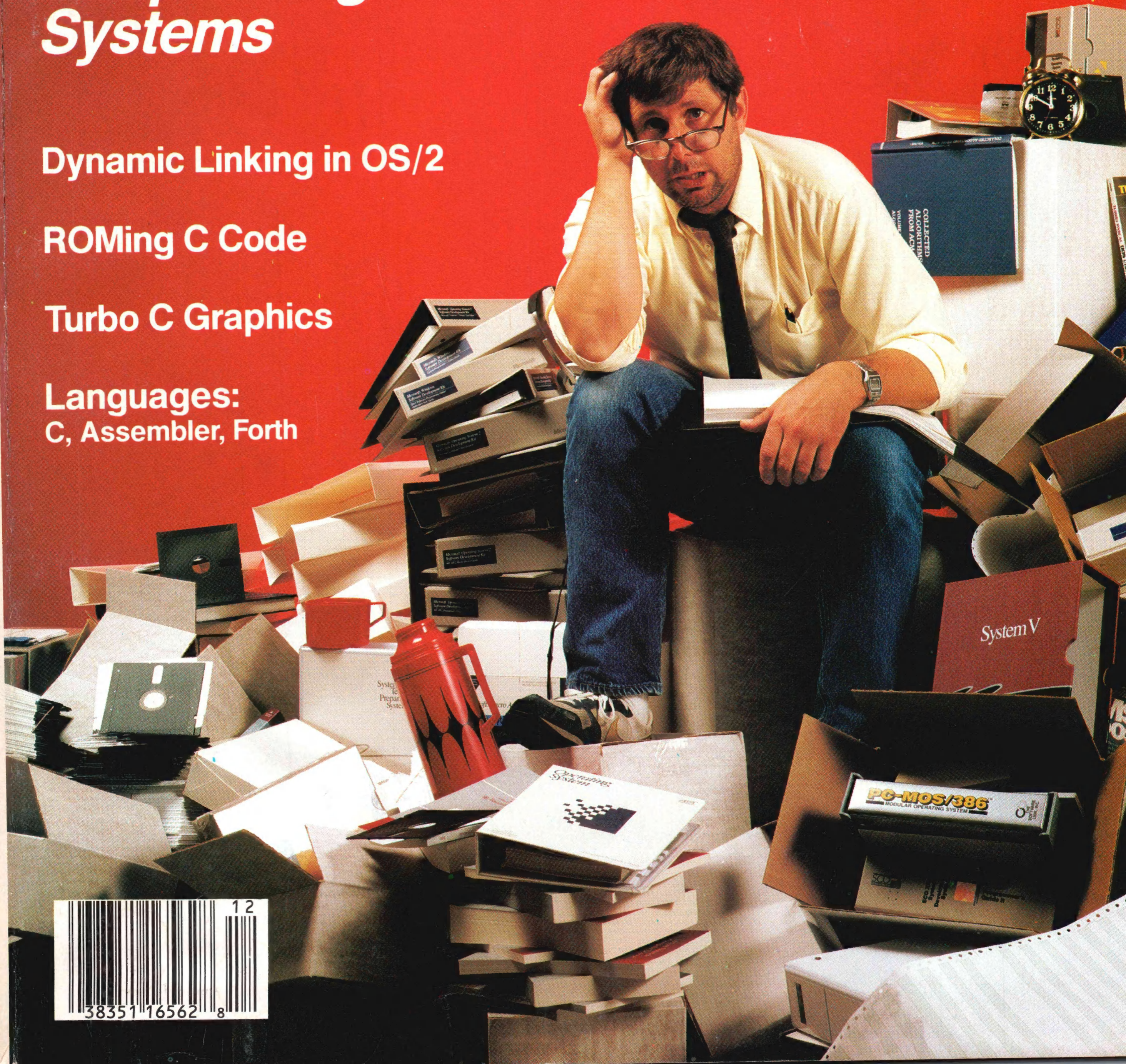
Making Sense of Operating Systems

Dynamic Linking in OS/2

ROMing C Code

Turbo C Graphics

Languages:
C, Assembler, Forth



#134 DECEMBER 1987

...han ever before!

4.0 uses logical units for separate compilation

Pascal 4.0 lets you break up the code gang into "units," or "chunks." These logical modules can be worked with swiftly and separately—so that an error in one module is seeable and fixable, and you're not sent through all your code to find one error. Compiling and linking these separate units happens in a flash because your compiling horsepower is better than 27,000 lines a minute.* And 4.0 also includes an automatic project Make.

4.0's cursor automatically lands on any trouble spot

4.0's interactive error detection and location means that the cursor automatically lands where the error is. While you're compiling or running a program, you get an error message at the top of your screen *and* the cursor flags the error's location for you.

4.0 gives you an integrated programming environment

4.0's integrated environment includes pull-down menus and a built-in editor. Your program output is automatically saved and shown in the output window. You can Scroll, Pan, or Page through all your output and know where everything is all the time. Given 4.0's integration, you can edit, compile, find and correct errors—all from inside the integrated development environment.

You'll never lose your mind, because 4.0 never loses your place

Whenever you re-load 4.0, it remembers what you and it were doing before you left. It puts you right back in the editor with the same file and in the same place as you were working last.

*Run on an 8 MHz IBM AT.

**If within 60 days of purchase this product does not perform in accordance with our claims, call our customer service department, and we will arrange a refund.

Please check box(es)	*Sugg. Retail*	*Upgrade Price†*	Serial No.
☐ Turbo Pascal 4.0 Compiler	$ 99.95	$ 39.95	______
☐ Turbo Pascal 4.0 Developer's Library	395.00	150.00	
(Includes Turbo Pascal Tutor and all Toolboxes; must be ordered with Compiler)			
☐ Turbo Pascal Tutor	69.95	19.95	______
☐ Turbo Pascal Database Toolbox	99.95	29.95	______
☐ Turbo Pascal Graphix Toolbox	99.95	29.95	______
☐ Turbo Pascal Editor Toolbox	99.95	29.95	______
☐ Turbo Pascal Numerical Methods Toolbox	99.95	29.95	______
☐ Turbo Pascal Gameworks	99.95	29.95	______

Total product amount $ ______
CA and MA residents add sales tax $ ______
Shipping and handling* $ ______
Total amount enclosed $ ______

Please specify diskette size: ☐ 5¼" ☐ 3½"   Payment: ☐ VISA ☐ MC ☐ Check ☐ Bank Draft

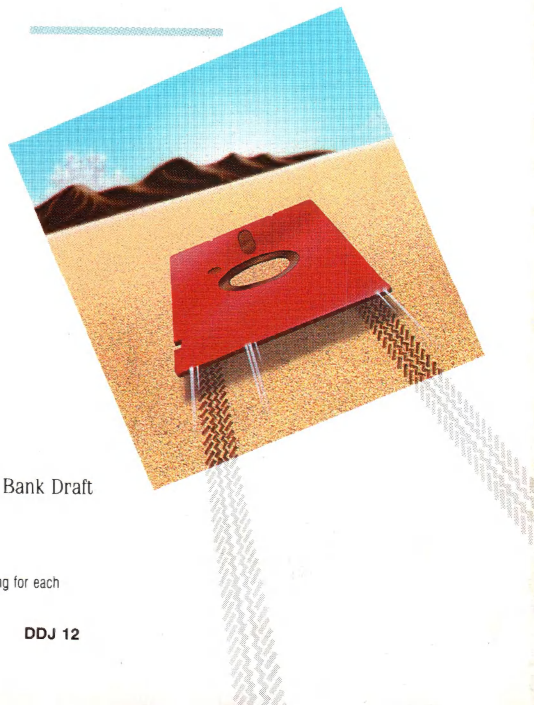
Credit card expiration date: ______/______

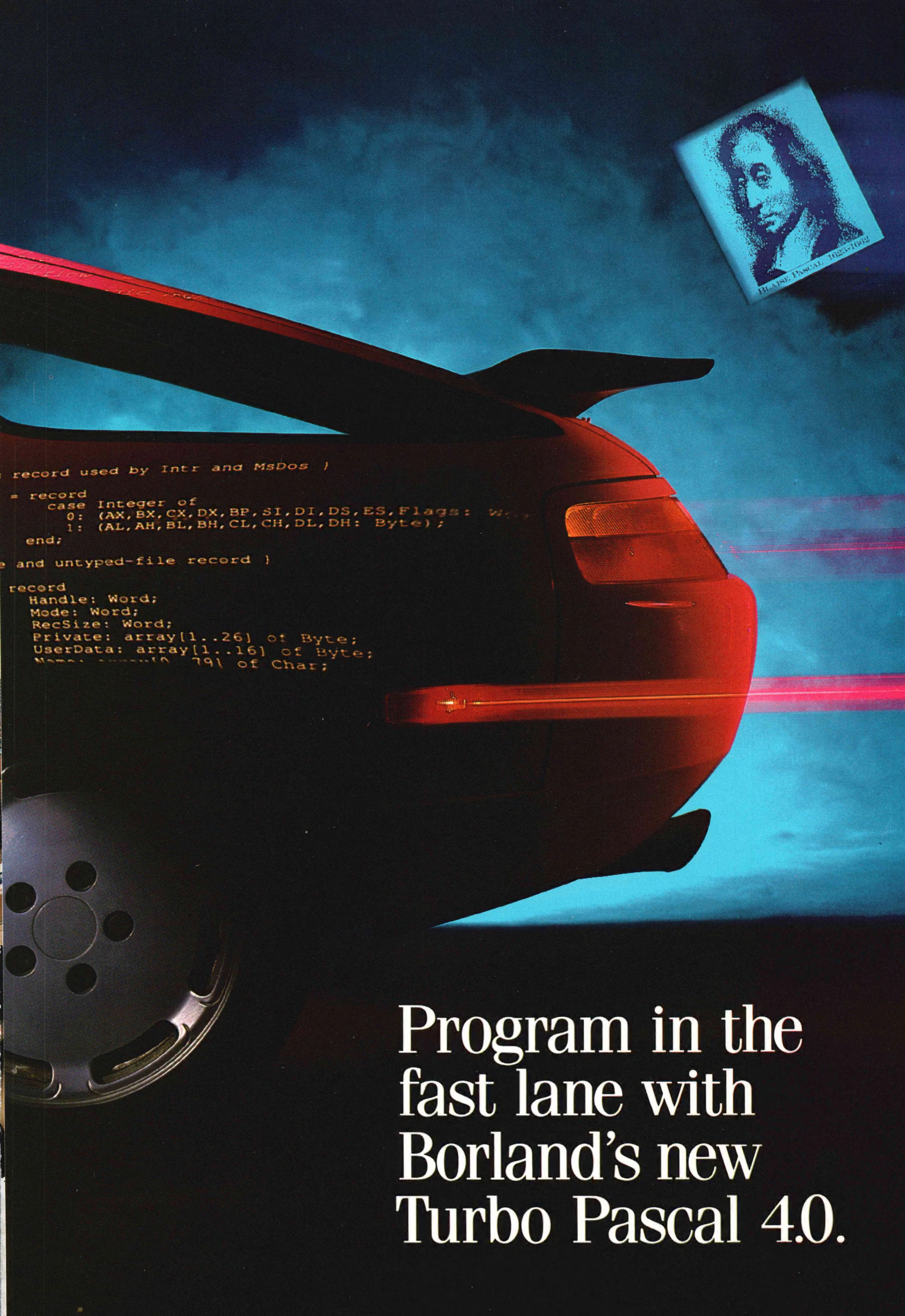
Card # |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

*In US please add $5 shipping for each product ordered or $15 for the Compiler and Developer's Library. Outside US please add $10 shipping and handling for each product ordered or $25 for the Compiler and Developer's Library.

†To qualify for the upgrade price you must give the serial number of the equivalent product you are upgrading.

DDJ 12





BLAISE PASCAL 1623-1662
record used by Intr and MsDos }
= record
case Integer of
0: (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:
1: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
end;
and untyped-file record }
record
Handle: Word;
Mode: Word;
RecSize: Word;
Private: array[1..26] of Byte;
UserData: array[1..16] of Byte;
79] of Char;
Program in the fast lane with Borland's new Turbo Pascal 4.0.

Now's the time for a *fast* decision: Upgrade now to 4.0!

Compatibility with Turbo Pascal 3.0

We've created 4.0 to be highly compatible with version 3.0 and included a conversion program and compatibility units to help you convert all your 3.0 programs to 4.0.

Highlights of Borland's new Turbo Pascal 4.0

- Compiles 27,000 lines per minute
- Supports >64K programs
- Uses units for separate compilation
- Integrated development environment
- Interactive error detection/location
- Includes a command line version of the compiler

4.0 also

- Saves output screen in a window
- Supports 25, 43 and 50 lines per screen
- Generates MAP files for debugging
- Has graph units including CGA, EGA, VGA, MCGA, 3270 PC, AT & T 6300 & Hercules support
- Supports extended data types (including word, long integers)
- Does smart linking
- Comes with a free revised MicroCalc spreadsheet source code

4.0 is all yours for only $99.95

Sieve (25 iterations)

	Turbo Pascal 4.0	Turbo Pascal 3.0
Size of Executable File	2224 bytes	11682 bytes
Execution speed	9.3 seconds	9.7 seconds

Sieve of Eratosthenes, run on an 8MHz IBM AT

Since the source file above is too small to indicate a difference in compilation speed we compiled our GOMOKU program from Turbo Gameworks to give you a true sense of how much faster 4.0 really is!

Compilation of GO.PAS (1006 lines)

	Turbo Pascal 4.0	Turbo Pascal 3.0
Compilation speed	2.2 seconds	3.6 seconds
Lines per minute	27,436	16,750

GO.PAS compiled on an 8 MHz IBM AT

60-Day Money-Back Guarantee**

BORLAND
INTERNATIONAL

For the dealer nearest you or to order call
(800) 543-7543.

BORLAND

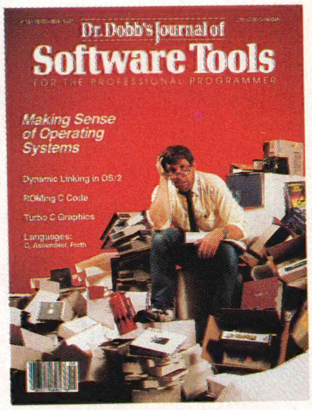
CONTENTS

ARTICLES

COLUMNS

FORUM

PROGRAMMER'S SERVICES



About the Cover

We'd like to be able to tell you that you could avoid all this OS documentation just by reading this single issue of DDJ. We'd like to, but we can't. The new trend in operating systems seems to be an entry requirement of at least six months' study.

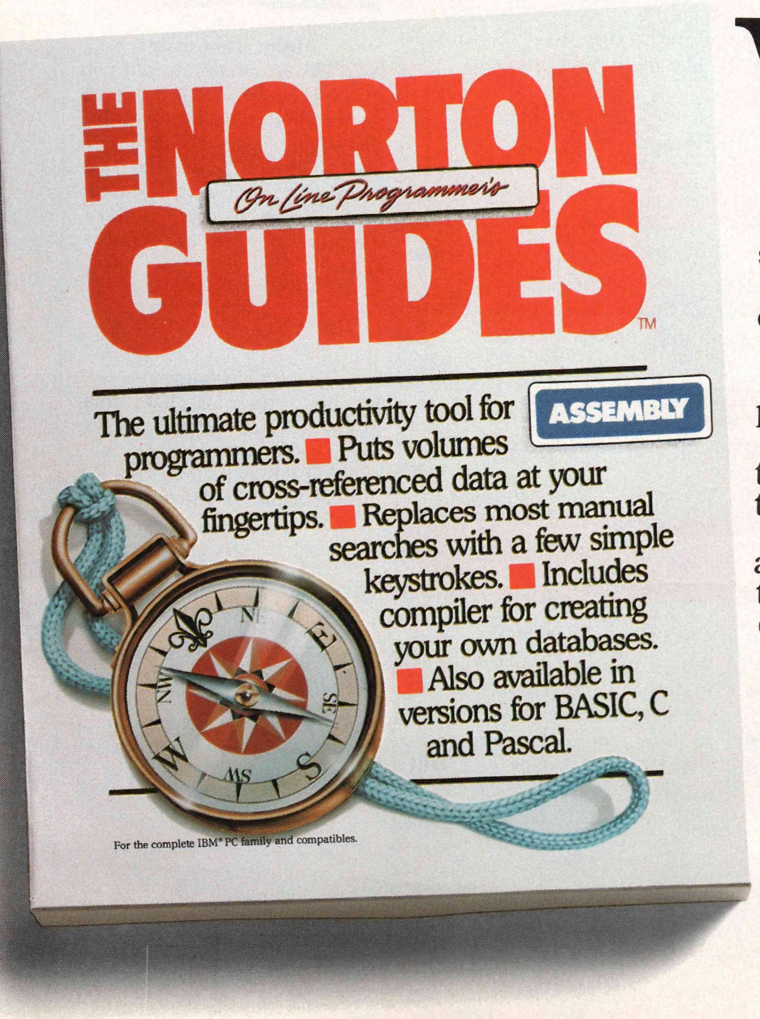
This Issue

It's the return of DDJ's infamous annual Operating Systems issue. So why are you reading this instead of the articles listed on the left?

Next Issue

January is DDJ's annual 68K issue, and we're starting the new year off right with a new column devoted to advanced Mac hacking, er, programming. Plus a couple of other surprises.

Peter Norton new programmi who hate



Nobody ever said programming PCs was supposed to be easy.

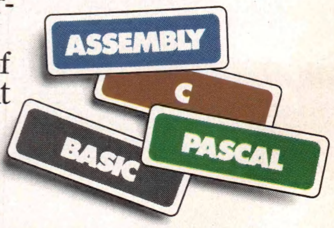
But does it have to be tedious and time-consuming, too?

Not any more.

Not since the arrival of the remarkable new program on the left.

Which is designed to save you most of the time you're currently spending searching through the books and manuals on the shelf above.

The Norton On-Line Programmer's Guides™ are a quartet of pop-up reference packages that do the same things in four different languages.



Each package consists of two parts: A memory-resident Instant Access™ program. And a comprehensive, cross-referenced database crammed with just about everything you need to know to program in your favorite language.

And when we say everything, we mean everything.

Everything from information about language

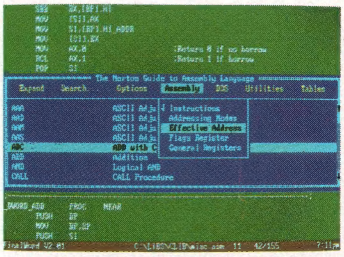
Designed for IBM® PC, PC-AT and DOS compatibles. Available at most software

announces a
ng tool for people
e manual labor.

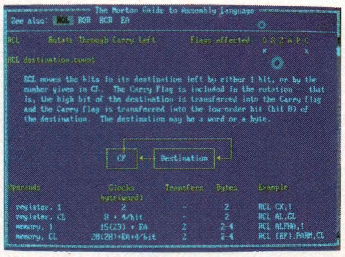
syntax to a variety of tables, including ASCII characters, line drawing characters, error messages, memory usage maps, important data structures and more.

How much more?

Well, the databases for BASIC, C and Pascal give you detailed listings of all built-in and library functions.



A Guides reference summary screen (shown in blue) pops up on top of the program you're working on (shown in green).



Summary data expands on command into extensive detail. And you can select from a wide variety of information.

While the Assembly database delivers a complete collection of DOS service calls, interrupts and ROM BIOS routines.

You can, of course, find most of this information in the books and manuals on our shelf.

But Peter Norton—who's written a few books himself—figured you'd rather have it on your screen.

In seconds.

In full-screen or moveable half-screen mode.

Popping up right next to your work. Right where you need it.

This, you're probably thinking, is precisely the kind of thinking that produced the classic Norton Utilities.™

And you're right.

But even Peter Norton can't think of everything.

Which is why there's a built-in compiler for creating databases of your own.

And why all Guides databases are compatible with the Instant Access program in your original package.

So you can add more languages without spending a lot more money.

To get more information, call your dealer. Or call Peter Norton at 213-453-2361.

And ask for some guidance.

Peter Norton
COMPUTING

dealers, or direct from Peter Norton Computing, Inc., 2210 Wilshire Blvd., #186, Santa Monica, CA 90403. 213-453-2361, Fax 213-453-6398, MCI Mail: PNCI © 1987 Peter Norton Computing

CIRCLE 103 ON READER SERVICE CARD

EDITORIAL

DOS Ex Machina

Any month now, we'll be greeted with the official debut of Microsoft's OS/2. It is, if you'll believe Bill Gates' patter, the operating system we've all been waiting for. At long last, we'll have multitasking and large program support. So why aren't we all more enthusiastic about it? Why is it that, as we get closer and closer to actually having OS/2, the alternatives begin to look so much better?

Some of this may be the result of the prolonged vaporware period. And some of it is no doubt the skepticism we normally feel at the introduction of any new software. In the case of OS/2, I think it's fair to say that the critics have had plenty to work with. There are all those minor technical points you've heard before: it's not thoroughly debugged yet; it's too fat; it's too slow; it's too expensive. Well, yes, true enough. But I'd like to remind everyone of a problem that dwarfs all those above.

Simply put, OS/2 is too late.

Note that I'm not talking about delays in shipping the product. Microsoft has been pretty good on meeting the ship dates for its OS/2 Developer Kits. No, what I'm talking about here is a critical marketing problem for OS/2: it's a dynamite product for a market that existed a couple of years ago.

The problem, as you've no doubt noticed, is that the world just didn't stop moving when the 80286 came along. As I write this, just weeks before Fall Comdex, there are several companies offering 80386 motherboard upgrades for XT owners. Intel is scheduled to announce their new Inboard 386/PC designed to fit into a normal PC. No toggle switches, no software tweaking. You just drop it in and your PC is an order of magnitude faster. (The RAM on the 386 card is well-used: some for faster ROM BIOS access, and some for caching the hard disk.) And this for a price cheaper than you're used to seeing AT-clones sell for.

So, given that 386 revolution is well under way, you have to wonder about the leadership demonstrated in Microsoft's introducing a 286-bound operating system when everyone is gearing up for the 386 revolution. Probably the most telling indictment of Microsoft's strategy is the market's reaction to the introduction of its own Windows/386. Forecasting the success and failure of software is always a tricky matter, but it seems reasonable to presume that Windows/386 will be vastly more successful than OS/2 over the short run. Indeed, the release of Windows/386 before OS/2 with its Presentation Manager could be seen as Microsoft's tacit admission that an OS/2 tomorrow is no match for a DesqView today.

I mention all this because in the months to come, *DDJ* will be spending a great deal of space exploring the 386 universe. We'll be investigating a variety of operating systems and environments. And we'll be covering OS/2, of course. It's just sad to realize that our OS/2 coverage, even as it begins, may be as dated in another year as CP/M Plus coverage is today.

Tyler Sperry

Tyler Sperry
editor

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Editorial
Editor-in-Chief *Michael Swaine*
Editor *Tyler Sperry*
Managing Editor *Vince Leone*
Associate Editor *Ron Copeland*
Assistant Editor *Sara Noah Ruddy*
Technical Editors *Allen Holub*, *Richard Relph*
Contributing Editors *Kent Porter*, *Namir Shammas*, *Ernest R. Tello*
Copy Editor *Rhoda Simmons*
Production
Director Art/Production *Larry L. Clay*
Art Director *Michael Hollister*
Assoc. Art Director *Joe Sikoryak*
Technical Illustrator *Barbara Mautz*
Typesetter *Mary E. Lopez*
Cover Photographer *Michael Carr*
Circulation
Circulation Director *Maureen Kaminski*
Fulfillment Coordinator *Francesca Martin*
Book Marketing Mgr. *Jane Sharninghouse*
Subscription Supervisor *Kathleen Shay*
Newsstand Sales Coordinator *Larry Hupman*
Administration
Finance Director *Kate Wheat*
Business Manager *Betty Trickett*
Accounts Payable Supv. *Mayda Lopez-Quintana*
Accts. Receivable Supv. *Laura DiLazzaro*
Advertising Director
Ferris Ferdon (415) 366-3600
Marketing Mgr. *Michael Wiener*
Trafficking Coordinator *Patricia Albert*
Account Managers *see page 129*
Associate Publisher
Michael Swaine
Assistant *Sara Noah Ruddy*

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points. *DDJ* is published under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

Article Submissions: Send manuscripts and disk (with article and listings) to the Associate Editor.

***DDJ* on CompuServe:** Type GO DDJ

Address Correction Request: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 088-3076**

Customer Service: For subscription problems call: outside CA (800) 321-3333; in CA (619) 485-9623 or 566-6947. For book/software order problems call (415) 366-3600.

Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $27 per year airmail or $10 per year surface. All other countries add $27 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX 620430 (WUI).

M&T Publishing Inc.
Chairman of the Board *Otmar Weber*
Director *C.F. von Quadt*
President and Publisher *Laird Foshay*

Our thanks to NASA for supplying this computer enhanced ultraviolet photo taken by Skylab IV of a solar prominence reaching out 350,000 miles above the sun's surface

Genius Begins With A Great Idea ...

But The Idea Is Just The Beginning

What follows is the time consuming task of giving form and function to the idea.

That's why we concentrate on building into our software development systems functions and features that help you develop your software ideas in less time and with less effort.

We've started 1987 by releasing new versions of our MS-DOS, Macintosh, Amiga, ROM, and Apple // C development systems. Each system is packed with new features, impressive performance, and a little bit more genius.

FREE 2-DAY DELIVERY!

Aztec C86 4.1

New PC/MS-DOS • CP/M-86 • ROM

Superior performance, a powerful new array of features and utilities, and pricing that is unmatched make the new Aztec C86 the first choice of serious software developers.

Aztec C86-p Professional System $199
• optimized C with near, far, huge, small, and large memory + Inline assembler + Inline 8087/80287 + ANSI support + Fast Float (32 bit) + optimization options • Manx Aztec 8086/80x86 macro assembler • Aztec overlay linker (large/small model) • source level debugger • object librarian • 3.x file sharing & locking • comprehensive libraries of UNIX, DOS, Screen, Graphics, and special run time routines.

Aztec C86-d Developer System $299
• includes all of Aztec C86-p • Unix utilities make, diff, grep • vi editor • 6 + memory models • Profiler.

Aztec C86-c Commercial System. $499
• includes all of Aztec C86-d • Source for library routines • ROM Support • CP/M-86 support • One year of updates.

Aztec C86 Third Party Software

A large array of support software is available for Aztec C86. Call or write for information. The following is a list of the most requested products: • Essential Graphics • C Utility Library • Curses • Greenleaf Communication, General, and Data Window • Halo • Panel+ • PC-lint • PforCe • Pre-C • Windows for C • Windows for Data • C terp • db_Vista • db-Query • Phact • Plink-86 Plus • c-tree • r-tree • Pmate

CP/M • TRS-80 • 8080/Z80 ROM

C compiler, 8080/Z80 assembler, linker, librarian, UNIX libraries, and specialized utilities.

Aztec C II-c (CP/M-80 & ROM). $349
Aztec CII-d (CP/M-80) $199
Aztec C80 (TRS-80 3&4) $199

Aztec C68k/Am 3.4

New Amiga Release

Amiga user groups across the USA voted Aztec C68k/Am release 3.3 the best Software Development System for the Amiga. Release 3.4 is more impressive.

Aztec C68k/Am-p Professional $199
A price/feature/performance miracle. System includes: optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Amiga libraries • examples.

Aztec C68k/Am-d Developer $299
The best of Manx, Amiga, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep and vi.

Aztec C68k/Am-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

FREE 2-DAY DELIVERY!

Aztec C68k/Mac 3.4

New Macintosh Release

For code quality, reliability, and solid professional features, Aztec C for the Macintosh is unbeatable. This new release includes features and functions not found in any other Macintosh C development system.

Aztec C68k/Mac-p Professional $199
• optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Macintosh libraries • examples.

Aztec C68k/Mac-d Developer. $299
The best of Manx, Macintosh, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep • vi editor.

Aztec C68k/Mac-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

Aztec C65

New ProDOS Release

Aztec C65 is the only commercial quality C compiler for the Apple II. Aztec C65 includes C compiler, 6502/65C02 assembler, linker, library utility, UNIX libraries, special purpose libraries, shell development environment, and more. An impressive system.

Aztec C65-c Commercial $299
• runs under ProDOS • code for ProDOS or DOS 3.3

Aztec C65-d Developer. $199
• runs under DOS 3.3 • code for DOS 3.3

Aztec ROM Systems

6502/65C02 • 8080/Z80 • 8086/80x86 • 680x0

An IBM or Macintosh is not only a less expensive way to develop ROM code, it's better. Targets include the 6502/65C02, 8080/Z80, 8086/80x86, and 680x0.

Aztec C has an excellent reputation for producing compact high performance code. Our systems for under $1,000 outperform systems priced at over $10,000.

Initial Host Plus Target $750
Additional Targets $500
ROM Support Package $500

Vax, Sun, PDP-11 ROM HOSTS

Call for information on Vax, PDP-11, Sun and other host environments.

C' Prime

PC/MS-DOS • Macintosh
Apple II • TRS-80 • CP/M

These C development systems are unbeatable for the price. They are earlier versions of Aztec C that originally sold for as much as $500. Each system includes C compiler, assembler, linker, librarian, UNIX routines, and more. Special discounts are available for use as course material.

C' Prime $75

Aztec Cross Development Systems

Most Aztec C systems are available as cross development systems. Hosts include: PC/MS-DOS, Macintosh, CP/M, Vax, PDP-11, Sun, and others. Call for information and pricing.

How To Become An Aztec C User

To become a user call 800-221-0440. From NJ or international locations call 201-542-2121. Telex: 4995812 or FAX: 201-542-8386. C.O.D., VISA, MasterCard, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Aztec C is available directly from Manx and from technically oriented computer and software stores. Aztec Systems bought directly from Manx have a 30 day satisfaction guarantee.

Most systems are upgradable by paying the difference in price plus $10. Site licenses, OEM, educational, and multiple copy discounts are available.

MANX

To order or for more information call today.

1-800-221-0440

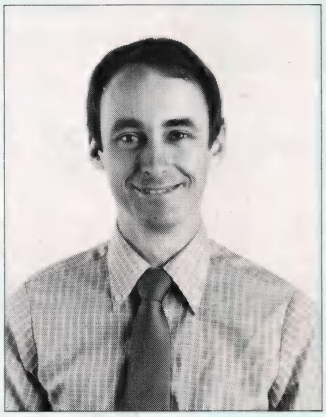
In NJ or international call (201) 542-2121 • TELEX: 4995812

CIRCLE 104 ON READER SERVICE CARD

Manx Software Systems
1 Industrial Way, Eatontown, NJ 07724

MS is a registered TM of Microsoft, Inc., CP/M TM DRI, HALO TM Media Cybernetics, PANEL TM Roundhill Computer Systems, Ltd., PHACT TM PHACT Assoc., PRE-C, Plink-86, Plink-86 +, P-Force TM Phoenix, db Vista TM Raima Corp., C-terp, PC-lint, TM Gimpel Software, C-tree TM Faircom, Inc., Windows for C, Windows for DATA TM Creative Solutions, Apple II, Macintosh TM Apple, Inc., TRS-80 TM Radio Shack, Amiga TM Commodore Int'l., Unix TM AT&T, Vax TM DEC, Aztec TM Manx Software Systems.

RUNNING LIGHT



It's been a crazy month, no doubt about it. I'd tell you all about it, except you'd probably be bored with the details, and as the old joke goes, I don't want to talk about it. Suffice to say that this month has seen the departure of Vince Leone, our long-suffering Managing Editor, and the usual mad rush to get the magazine out the door has been unusually mad.

This month has also seen an abnormal mix of last minute code changes and too little space in the magazine. The code for Rick Naro's LOCATE utility, for example, was updated for MASM version 5.0, but unfortunately we've had to continue the listings into next month. That code will be joined by some last-minute additions to the articles by Dave Cortesi and Martin Tracy. Those of you who use CompuServe won't be affected by most of this madness since we'll be updating the code before it goes online. And before it goes onto the listings disk.

One last note before I go: the gremlins obliterated my CompuServe number last month, but you can still reach me as 76703,4266. If you're doing neat things in Object-oriented programming or AI, now's the time to pitch me an article.

Tyler Sperry

Tyler Sperry
editor

STATEMENT OF OWNERSHIP, MANAGEMENT, AND CIRCULATION
(Act of August 12, 1970, Section 3685, Title 39, United States Code)

1. Title of Publication: Dr. Dobb's Journal of Software Tools, Publication No. 08883076.
2. Date of Filing: October 10, 1987.
3. Frequency of Issue: Monthly (12 issues, $29.97).
4. Location of Known Office of Publication: 501 Galveston Dr., Redwood City, CA 94063.
5. Location of Headquarters of General Business Offices of the Publishers: 501 Galveston Dr., Redwood City, CA 94063.
6. Names and Addresses of Publisher, Editor, and Managing Editor: Publisher, Laird Foshay, 501 Galveston Dr., Redwood City, CA 94063. Editor, Michael Swaine, 501 Galveston Dr., Redwood City, CA 94063. Managing Editor, Vince Leone, 501 Galveston Dr., Redwood City, CA 94063.
7. Owner: M&T Publishing, 501 Galveston Dr., Redwood City, CA 94063.
8. Known Bondholders, Mortgages, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages or Other Securities: Markt & Technik, Hans-Pinsel-Strasse 2, 8013 Haar bei Munich W. Germany.
9. Extent and Nature of Circulation:
   A. Total Number of copies printed. Average number of copies each issue during the preceding 12 months: 70,701. Actual number of copies of single issue published nearest to filing date: 76,069.
   B. Paid Circulation. 1. Sales through dealers and carriers, street vendors, and counter sales. Average number of copies each issue during preceding 12 months: 14,727. Actual number of copies of single issue published nearest to filing date: 17,258. 2. Mail subscriptions. Average number of copies each issue during preceding 12 months: 41,486. Actual number of copies of single issue published nearest to filing date: 42,697.
   C. Total Paid Circulation. Average number of copies each issue during preceding 12 months: 56,213. Actual number of copies of single issue published nearest to filing date: 59,955.
   D. Free distribution by mail, carrier, or other means, samples, complimentary, and other free copies. Average number of copies each issue during preceding 12 months: 2,266. Actual number of copies of single issue published nearest to filing date: 1,900
   E. Total distribution. Average number of copies each issue during preceding 12 months: 58,479. Actual number of copies of single issue published nearest to the filing date: 61,855.
   F. Copies not distributed. 1. Office use, left over, unaccounted, spoiled after printing. Average number of copies each issue during preceding 12 months: 650. Actual number of copies of single issue published nearest to filing date: 655. Returns from News Agents. Average number of copies each issue during preceding 12 months: 11,572. Actual number copies of single issue published nearest to filing date: 13,559.
   G. Total. Average number of copies each issue during preceding 12 months: 70,701. Actual number of copies of single issue published nearest to filing date: 76,069.

I certify that the statements made by me above are correct and complete.

Laird Foshay, Publisher

ARCHIVES

Ten Years Ago in *DDJ*

"DALLAS, Texas—The first flexible disk drive for $5^{1}/_{4}''$ diskettes to offer double density recording of 250,000 bytes on each side of a diskette, was introduced at the National Computer Conference here today by the Pertec Division of Pertec Computer Corporation." *News release received Jun 18, 1977,* DDJ, *November/December 1977.*

Kicking the 8080 Habit

"Maybe it's just my imagination, but it seems that a lot of people aren't utilizing the Z-80 to its fullest. Everyone is so used to writing code for the 8080 that they don't seem to bother upgrading their software when they upgrade their CPU. . .

"I would like to see you guys. . . explain all the nifty Z-80 tricks. I know I can't be the only one that is stuck in the rut of 8080 code. (Please!! Don't tell me I swapped my CPU board JUST for speed—the software potential is fantastic.)" *Letters to the Editor,* DDJ, *November/December 1977.*

Exec with Extreme Prejudice

"In a multi-tasking environment such as we expect to see in MS-DOS 3.0, this function (function 4BH - EXEC) will be even more useful and will undoubtedly be elaborated with several additional features. Under such an operating system, a parent task can 'spawn' any number of child tasks, which can execute concurrently and asynchronously, and communicate by means of queues, semaphores, and pipes.

"Well, you say, pie in the sky is all very nice, but why is the EXEC function getting so much attention in this magazine column? The answer is, of course, that when I tried to actually use the function I ran into any number of glitches and hazy spots in the documentation." *Ray Duncan, "16-bit Software Toolbox,"* DDJ, *December 1983.*

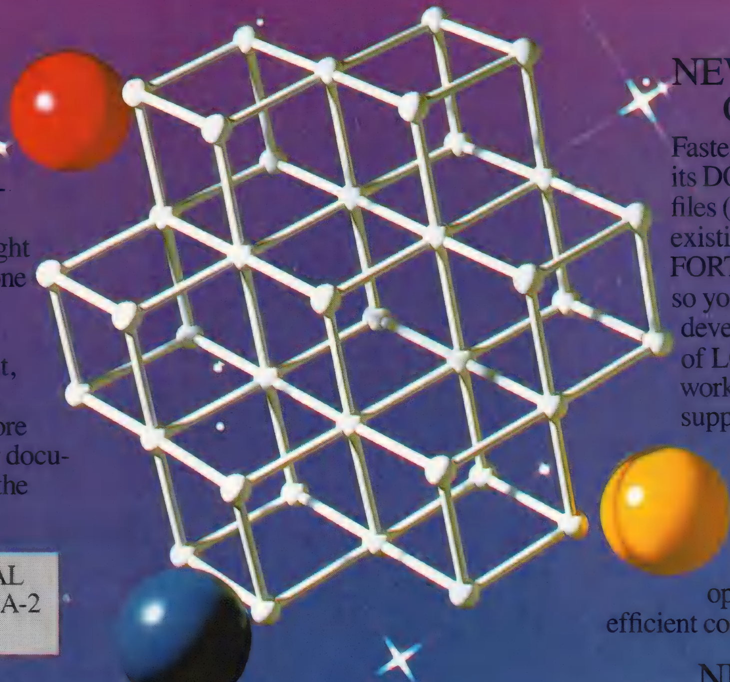
Encryption

"A small but vital piece of hardware containing a microelectronic chip only 1 cm square has been tested and validated at the Commerce Department's National Bureau of Standards (NBS)—marking the first NBS validation of a commercial implementation of the Federal Data Encryption Standard published early this year." *News release received October 31, 1977,* DDJ, *November/December 1977.*

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia
Running Light Without Overbyte

SOFTWARE ENGINEERING COMES OF AGE.

ANNOUNCING LOGITECH MODULA-2 VERSION 3.0



Modula-2 is the language of choice for modern software engineering, and LOGITECH Modula-2 is the most powerful implementation available for the PC. The right language and the right tools have come together in one superior product. Whether you're working on a small program or a complex project, with LOGITECH Modula-2 Version 3.0 you can write more reliable, maintainable, better documented code in a fraction of the time at a fraction of the cost.

FREE TURBO PASCAL TO LOGITECH MODULA-2 TRANSLATOR

NEW, IMPROVED DEBUGGERS

Time gained with a fast compiler can be lost at debug time without the right debugging tools. With the powerful Logitech Modula-2 Debuggers you can debug your code *fast,* and dramatically improve your overall project throughput.

The Post Mortem Debugger analyzes the status of a program after it has terminated while the dynamic, Run Time Debugger monitors the execution of a program with user-defined breakpoints. With their new, mouse based, multiple-window user interface these powerful debugging tools are a pleasure to use.

NEW, INTELLIGENT LINKER

Links only those routines from a particular module that you need, so you eliminate unreferenced routines and produce smaller, more compact executable files.

NEW, IMPROVED COMPILER

Faster and more flexible. Now its DOS linker compatible object files (.OBJ) can be linked with existing libraries in C, PASCAL, FORTRAN and ASSEMBLER — so you can build on previous development and put the power of LOGITECH Modula-2 to work for you right now. Fully supports Wirth's latest language definition, including LONGINT and LONGSET, which provides large set support including SET of CHAR. Provides optimization for tighter, more efficient code generation.

NEW EDITOR

Our new, mouse based editor is fully integrated, easy to learn, fast and easy to use, and very customizable. Its multiple, overlapping windows and color support make it easy to manage parts of one file or several files on the screen at one time. You'll love using it — with or without a mouse.

Call for information about our VAX/VMS version, Site License, University Discounts, Dealer & Distributor pricing.

To place an order call toll-free:
800-231-7717
In California:
800-552-8885

☐ LOGITECH Modula-2 V. 3.0 Compiler Pack — $99
Compiler in overlay and fully linked form, Linkable Library, Post Mortem Debugger, Point Editor

☐ LOGITECH Modula-2 V. 3.0 Toolkit — $169
Library sources, Linker, Run Time Debugger, MAKE, Decoder, Version, XRef, Formatter

☐ LOGITECH Modula-2 V. 3.0 Development System — $249
Compiler Pack plus Toolkit

☐ Turbo Pascal to Modula-2 Translator — FREE
With Compiler Pack or Development System

☐ Window Package — $49
Build true windowing into your Modula-2 code.

☐ Upgrade Package
Call LOGITECH for information or to receive an order form.

Add $6.50 for shipping and handling. California residents add applicable sales tax. Prices valid in U.S. only. Total Enclosed $ ________

☐ VISA ☐ MasterCard ☐ Check Enclosed

Card Number ________ Expiration Date ________

Signature ________

Name ________

Address ________

City ________ State ________

Zip ________ Phone ________

LOGITECH

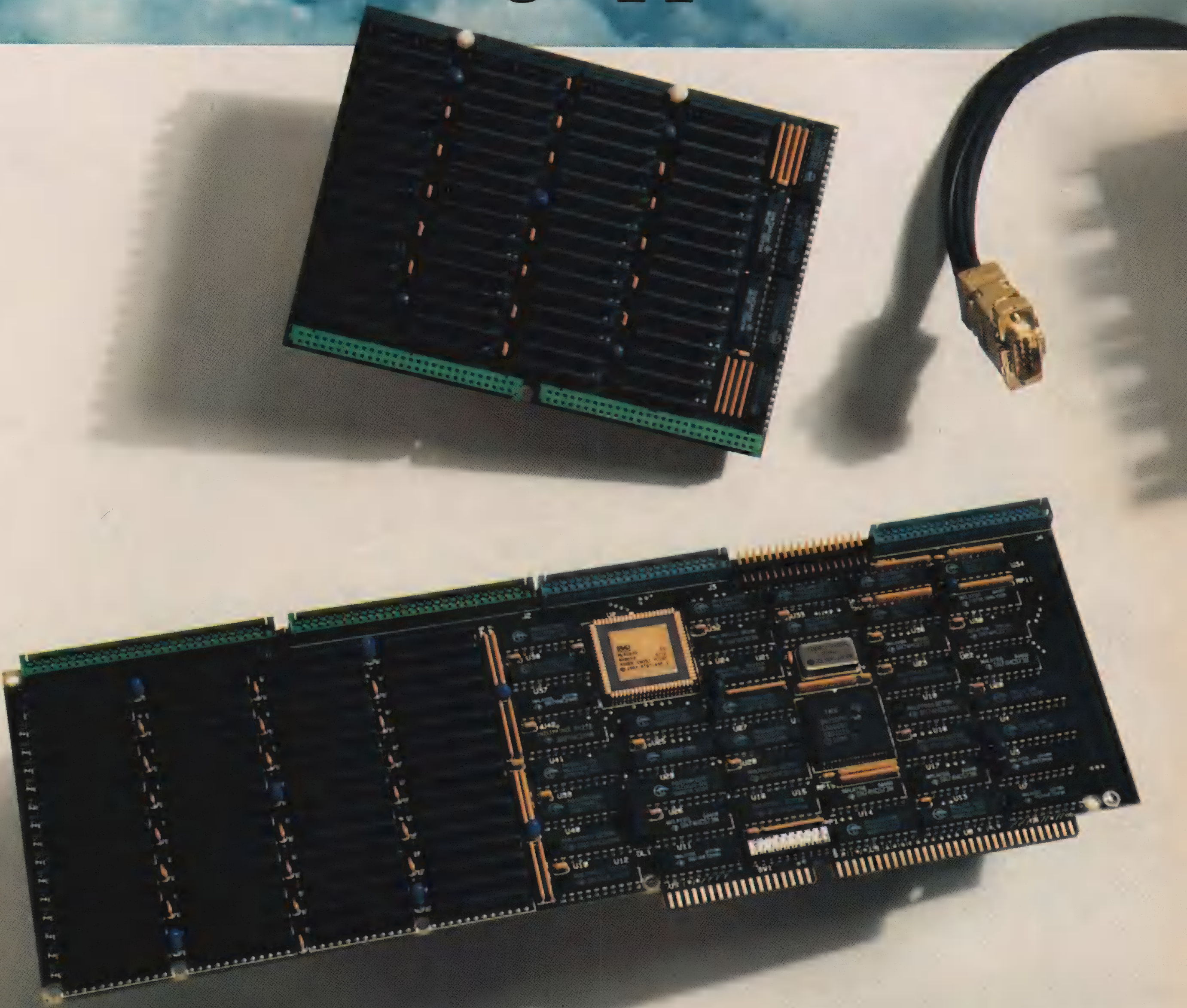
LOGITECH, Inc.
6505 Kaiser Drive, Fremont, CA 94555
Tel: 415-795-8500

In Europe: LOGITECH, Switzerland
Tel: 41-21-87-9656 Telex 458 217 Tech Ch

In the United Kingdom: LOGITECH, U.K.
Tel: 44908-368071 Fax: 44908-71751

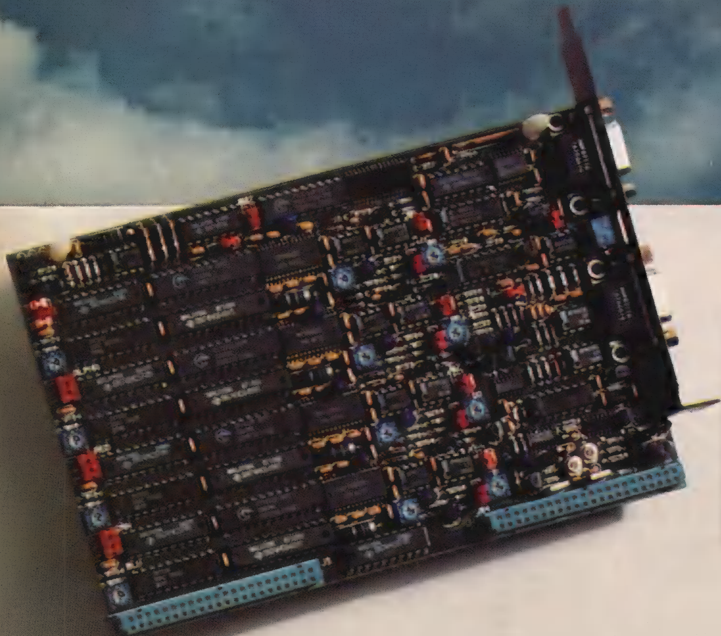
CIRCLE 105 ON READER SERVICE CARD

Now Taking Applications.

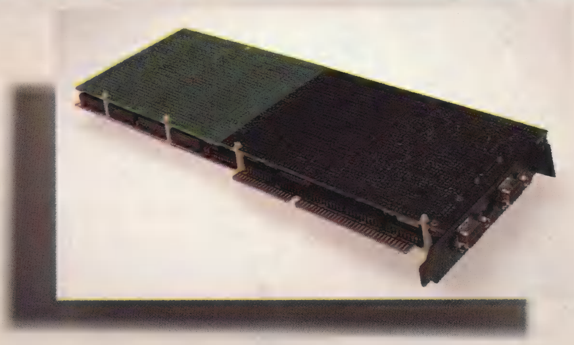


Take a look at the specs on VISTA™, a good look. Notice the processing, programming, and video capabilities? Now think real hard about what *you* could do with the power of VISTA and a microcomputer. Incorporate it with your system to create a digital pre-press proofing station for publishing. Design a graphics workstation which outputs both colorful hi-resolution slides and broadcast-quality animated images. Construct a CAD system which merges computer generated images with real-life backdrops for architecture, packaging or other industries. And, after you've brainstormed your way to new horizons of videographics possibilities, get your own VISTA and start working.

Introducing VISTA™ Videographics.



VISTA is a powerful single-slot videographics adapter which captures and displays video signals in real time.



Let's Get Specific.

We knew you couldn't resist seeing the facts, and frankly, our engineers wouldn't have it any other way. Here is an overview of VISTA's key features.

FEATURES:

- 4Mbytes of Video RAM on-board
- Texas Instruments' TMS 34010 GSP
- Flexible, programmable resolutions
- NTSC and PAL compatible
- Four 8-bit channels for real-time capture
- Fully integrated genlock
- Processor memory expandable in 2Mbyte increments to 12Mbytes
- Four 2K x 8-bit CMOS static RAM LUTs
- Display can be color-mapped, RGB, or a versatile combination of both
- Interlaced and non-interlaced display
- Binary and fractional programmable zoom capability, creates horizontal and vertical magnify or minify
- Smooth horizontal and vertical programmable panning, includes wrap-around and split screen
- Suggested Retail Price: $5995.

ADDRESSABLE RESOLUTIONS:

32 bits/pixel	16 bits/pixel	8 bits/pixel
1024x1024	2048x1024	4096x1024
512x2048	1024x2048	2048x2048
256x4096	512x4096	1024x4096

CAPTURE RESOLUTIONS:*

NTSC	PAL
(RS-170A)	(CCIR-624)
756x486	738x576
604x486	590x576
504x486	492x576
432x486	422x576

Resolutions are programmable; these are nominal ones for interlaced NTSC and PAL compatible.

DISPLAY RESOLUTIONS:*

NTSC	PAL	Interlaced	Non-Interlaced
(RS-170A)	(CCIR-624)		
1512x486	1476x576	1024x768	768x576
1008x486	984x576	(60 Hz)	(50 Hz)
756x486	738x576		
604x486	590x576	768x768	756x486
504x486	492x576	(80 Hz)	(60 Hz)

Resolutions are programmable; these are nominal ones.

COMPUTER REQUIREMENTS:

Host Type:	IBM PC AT and 100% Compatibles, Compaq 386, Apollo DN 3000-single-slot board
Data Bus:	16-bit or 8-bit (self-configuring)
Bus Clock:	6MHz to 12MHz
Power Consumption:	15 Watts

It's So Flexible, We've Added Support.

With its Texas Instruments TMS 34010 graphics processor, large quantity of video memory, and proprietary video cross-point, VISTA can be programmed for an array of powerful market-specific videographic applications. To help you maximize VISTA's potential, Truevision offers a range of C-language programming tools for developers. And when your system is market-ready, we'll support your marketing efforts with our *TRUEVISION SOFTWARE CATALOG*, *TRUEVISION NEWS*, and *THE PULSE*.

We're For Higher

Resolution…Power…Flexibility… Quality. Join the many key manufacturers and developers already working with the state of the videographics art, VISTA. Call us at **800/858-TRUE** for more information on the VISTA Developer's Program. We're ready to take your application today.

AT&T
Electronic Photography and Imaging Center
7351 Shadeland Station, Suite 100
Indianapolis, IN 46256
800/858-TRUE

AT&T | TRUEVISION®

CIRCLE 106 ON READER SERVICE CARD

IBM is a registered trademark of International Business Machines Corp. **International Inquiries:** contact *Techexport* at 617/890-6507 (USA), or London at 44-1-991-0121. In Italy, contact *SIRIO Informatica* at 39-2-301-0051. Suggested retail price is US domestic price.

LETTERS

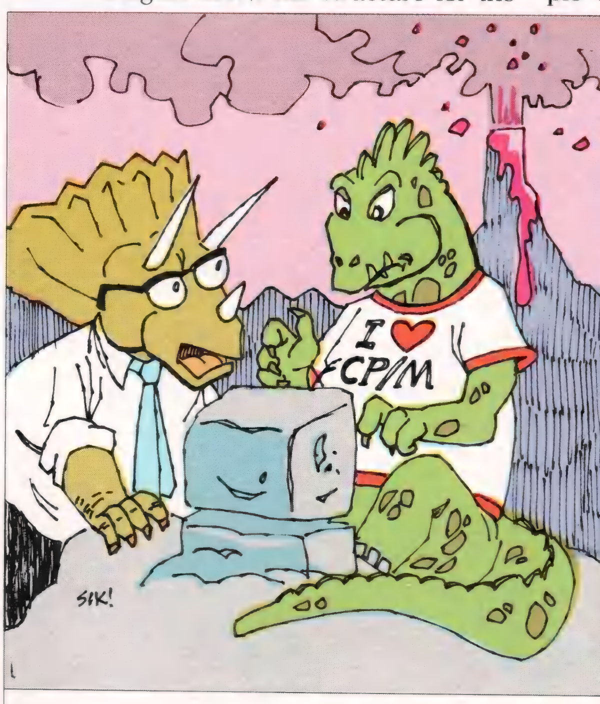
Stone Age Software

Dear *DDJ*,

Thank you for an excellent editorial in the September 1987 issue. The point Tyler Sperry makes about the crippling of potentially powerful machines is a sore point with me also.

I work for a large computer manufacturer and have seen for myself how these limiting factors get incorporated into machines, not only in software but also in hardware. I don't see an immediate solution but, like you, I'll keep screaming for the liberation of these systems from stupid design flaws.

Les J. Record
1201 East Mesa Pk. Dr.
Round Rock, TX 78664



Hurry up Jennings—our very existence may depend on it!

Dear *DDJ*,

I don't think there is any reason to complain about the architecture of the 80286, at least not about the inability to switch back to real mode. This is like building a V30-based computer to run CP/M Plus and then cursing at those crazy 16-bit data and I/O buses that make designing a good 8-bit system such a mess.

Remember, the 8086/8088 didn't even have an 8080 emulation mode, so why blame the designers of the 80286 for including an 8086 emulation mode? It is there just to give an upward migration path, not to be frantically switched on and off.

Let's face it, it's not the chip that is Stone Age but the software. Switching back from protected mode itself is Stone Age, not the messy way it has to be done. And AboveBoard emulators are a Stone Age way of reducing a 16-megabyte address space to an 8-megabyte collection of memory chunks—and there are people proud of that!

The cry for hardware compatibility—which means hardware-dependent software (which equals incompatible software)—has made the transition from today's status quo far more difficult than the change from 8-bit CP/M to 16-bit MS-DOS has been. It's like using metal tools to make better stone axes. It will take true Bronze Age men to change things.

Jost Riedel
Am Reservoir 2
P.O. Box 1141
D-3522 Bad Karlshafen 1
West Germany

Recursive/Iterative Trade-Off

Dear *DDJ*,

I really enjoyed the article on backtracking by Charles F. Bowman (August 1987). His structure for the puzzle-solving program, using recursive techniques, is nice and simple.

I just have one quibble with the article: In his discussion on ways to speed up the process, Bowman apologizes for his use of a recursive approach and states the conventional wisdom, "Recursive procedures are costly because of the considerable amount of overhead required for each successive call. Your program must save registers, store a return address, allocate local storage, and so on."

That statement surely sounds plausible. I've accepted it on faith for years. It was probably even true on most of the old mainframe computers, which is probably the context in which both Bowman and I heard it. But it may not be true in the context of microcomputers. All current microprocessors support very fast subroutine call/return mechanisms as well as stack push/pops. Because of these operations, sometimes recursive is better in all ways.

To test the truth of the recursive/iterative trade-off, I wrote an eight-queens problem in both forms (the recursive version is shown in Example 1, page 14). The program was written in Turbo Pascal for a PC clone. Much to my pleasant surprise, the recursive version turned out to be a full 40 percent faster than the nonrecursive form. It was also smaller, of course.

Another concern often voiced about recursive approaches is that of limited stack space. The idea is that, if your program has to go many levels of recursion, it may crash by overflowing the stack. To test that hypothesis, I ran the program shown in Example 2, page 14, again using Turbo Pascal. The program did indeed crash (gracefully) but at a level of more than 5,400 layers of recursion. That's 5,400 successive subroutine calls, folks! That should be enough for most of us!

So it appears that the conventional wisdom "recursion

How to tell the difference between DESQview™ 2.0 and any other environment.

Selecting DESQview, the environment of choice, can give you the productivity and power you crave, without the loss of your old programs and hardware. If you like your existing programs, want to use them together, transfer data between them, print, sort, communicate with or process-in-background, yet still have the need to keep in place your favorite PC(8088, 8086, 80286 or 80386), DESQview is the "proven true" multitasking, multiwindowing environment for you. Best of all, DESQview 2.0 is here now, with all the money saving, time saving, and productivity features that others can only promise for the all-too-distant future.

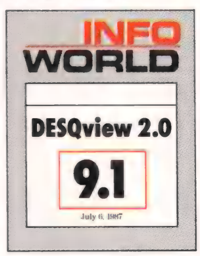
One picture is worth a thousand promises.

And with DESQview's new graphics enhancements for Hercules, CGA, EGA, and VGA, Version 2.0 still offers the same award winning and pioneering features for programs that earned DESQview its leadership, only now you can also run desktop publishing programs, CAD programs, even GEM-,™ Topview-,™ and Microsoft Windows-™ specific programs. In some cases you'll add as little as 10–40K to your system overhead. Now you can have multi-tasking, multi-windowing, break the 640K habit too and still get an auto dialer, macros, menus for DOS and, for advanced users, a new complete application programmer's interface capability. No wonder that over the years, and especially in recent months, DESQview, and now DESQview 2.0 have earned extravagant praise from some of the most respected magazines in the industry.

"Product of the Year" by readers vote in InfoWorld.

"Best PC Environment" by popular vote at Comdex Fall in PC Tech Journal's "System Builder" Contest.

"—I wouldn't want to run an IBM or compatible computer without DESQview"—InfoWorld, Michael Miller.



"A colossus among windowing environments"... "will run almost anything"—PC Week, Marvin Bryan.

"Windows, promises, but DESQview delivers"—MICROTIMES, Birell Walsh.

No other environment has consistently pioneered features, openness, and productivity. See for yourself. Send in the coupon. The possibilities are endless with DESQview 2.0.

Attention Programmers: For more information about Quarterdeck's API, and future 386 program extensions, call us today.

SYSTEM REQUIREMENTS
IBM Personal Computer and 100% compatibles (with 8086, 8088, 80286 or 80386 processors) with monochrome or color display; IBM Personal System/2 • Memory: 640K recommended; for DESQview itself 0-145K • Expanded Memory (Optional): expanded memory boards compatible with the Intel AboveBoard; enhanced expanded memory boards compatible with the AST RAMpage • Disk: Two diskette drives or one diskette drive and a hard disk • Graphics Card (Optional): Hercules, IBM Color/Graphics (CGA), IBM Enhanced Graphics (EGA), IBM Personal System/2 Advanced Graphics (VGA) • Mouse (Optional): Mouse Systems, Microsoft and compatibles • Modem for Auto-Dialer (Optional): Hayes or Compatible • Operating System: PC-DOS 2.0-3.3; MS-DOS2.0-3.2 • Software: Most PC-DOS and MS-DOS application programs; programs specific to TopView 1.1, GEM 1.1 and Microsoft Windows 1.03 • Media: DESQview 2.0 is available on either 5¼″ or 3½″ floppy diskettes

Rush me DESQview 2.0! Today! **DDJ12/87**

No. of Copies	Media 3½″/5¼″	Product	Retail Price ea.	Total
		DESQview 2.0	$129.95	$
		Shipping & Handling USA Outside USA	$ 5.00 $ 10.00	$
		Sales Tax (CA residents)	6.5%	$
Payment: ☐ Visa ☐ MC ☐ AMEX ☐ Check			Amount Enclosed	$

Credit Card: Valid Since ____/____ Expiration ____/____

Card Number: ________________

Credit Card Name ____________

Shipping Address ____________

City ____ State ____ Zip ____ Telephone ____

Mail to: Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405.
NOTE: If you own DESQview call us for a special upgrade offer, or send in your DESQview registration card. AST Special Edition users included.

DESQ view™
Quarterdeck

Quarterdeck Office Systems • 150 Pico Boulevard, Santa Monica, CA 90405 • (213) 392-9851

CIRCLE 107 ON READER SERVICE CARD

(continued from page 12)

```
{ A skeleton program for the eight-queens
problem. By altering the dimensions and the
routines fit, place, and unplace, it also
serves as a model for any other solution
involving backtracking. The program assumes a
 data array q[0..maxcount] containing the
values to be adjusted (row on the chessboard
in the case of the eight queens.  Three
procedures are assumed:
Function fit(l) compares the position of the
queen q[l] with all pieces placed so far,
and returns TRUE if there is no conflict.
Procedure place(l) records queen q[l] on the
board
Procedure unplace(l) removes queen q[l].
}

procedure Try(l: integer);
begin
   for q[l] := 0 to maxcount do begin
        if fit(l) then begin
             place(l);
             if l = max then ShowResult
             else try(l + 1);
             unplace(l);
        end;
   end;
end;

{ Main Program }

begin
   initialize;
   try(0);
end.
```

Example 1: *Eight queens problem in Pascal*

```
Program test;
procedure bump(n: integer);
begin
   writeln(n);
   bump(n + 1);
end

{ Main Program }

begin
   bump(0);
end.
```

Example 2: *Program to test levels of procedure nesting*

Bytes	8088 Clocks		
3	12+ea = 12+9 = 21	mov	ax,word ptr (bp).value
2	2	mov	bx,ax
3	12+ea = 12+9 = 21	mov	ax,word ptr (bp).value(2)
8	44		
3	12+ea = 12+9 = 21	mov	ax,word ptr (bp).value(2)
3	12+ea = 12+9 = 21	mov	bx,word ptr (bp).value
6	42		
3	24+ea = 24+9 = 33	les	bx,(bp).value
2	2	mov	ax,es
5	35		

Table 1: *Timings for Example 6, page 26, July 1987* DDJ

is costly" needs to be put to rest. In the modern world of micros, the simplest and most elegant solution may also be the smallest and fastest.

As an aside, the issue of fast call/return mechanisms should also cause us to take another hard look at assembly-language programming. The conventional picture of an assembly-language program is that of one long string of in-line code and macros. Many programmers tend to think this is necessary to achieve the speed you expect of a native-language program. But because of the support provided by the micro chips, the trade-offs of modularity vs. speed favor the former in assembly language, even more than in a higher-order language.

DDJ Forum User

Instruction Timings

Dear *DDJ*,

A reader called to point out an error in my article "8088 Assembly-Language Programming Techniques" (July 1987). It seems that the early versions of Intel's instruction timing tables implied that no cycles are used for effective address calculation when the AX register is used for moves to and from memory. In fact, this is the case only when using direct memory reference. For all other cases, AX is treated in the same way as any other register.

Table 1, left, gives the timings for my Example 6, on page 26 of the July issue. There is no doubt that the third method is best for real mode. Protected mode on the 80286, however, is another story.

Tom Disque
SAS Institute Inc.
P.O. Box 8000
SAS Circle
Cary, NC 27511-8000

DDJ

The Leaders Made PVCS The Leading Source Code Control System.

NOW VAX/VMS & MS-DOS Versions

When it comes to maintaining their most valuable asset, the leading software publishers rely on the POLYTRON Version Control System (PVCS). From accounting firms to airlines, the leading service companies depend on PVCS to maintain the integrity of their programs. Leading manufacturing companies use PVCS to maintain their state-of-the-art software. Leading high technology companies turn to PVCS to handle configuration management for software projects that represent an investment of hundreds of thousands of dollars. The largest aerospace companies and defense contractors use PVCS to maintain integrity of projects during development and after delivery of software. Independent programmers use PVCS to improve their productivity and software quality for themselves and their clients.

Simplify Configuration Management

When large and complex software programs are being developed on personal computers or VAX minicomputers, effective management of the revisions and versions becomes critical. PVCS simplifies this process and lets you effectively control the proliferation of code changes. We used UNIX SCCS and RCS as models. However, our own experience, and the input of hundreds of programmers and managers has enabled us to significantly improve upon these models.

PVCS provides many powerful functions including:

- Storage & Retrieval of multiple revisions of text.
- Maintenance of a complete history of changes.
- Maintenance of separate lines of development using branching.
- Merging simultaneous changes.
- Resolution of Access Conflicts.
- Modules can be retrieved by their own revision number, system version name, or specified date.
- Uses "reverse deltas" to rebuild a prior version making PVCS the fastest version control system over the project life cycle.
- Projects already under development or in the maintenance stage can be easily put under the control of PVCS.

Manages Development On Local Area Networks

Programming teams using Local Area Networks depend on PVCS to help the managers and team members work together. In fact, Novell and 3Com themselves depend on PVCS to manage the versions of their own network software products.

Supports MS-DOS and VAX/VMS Development

Now, companies that develop software on VAX systems running VMS can also use PVCS. And since the VMS and MS-DOS versions of PVCS use the same "logfile" format, you can easily develop software on PCs and maintain the code on the VAX or vice versa. The menu-driven, screen-oriented interface (and optional command-driven interface) makes it easy for programmers and librarians or administrators to use PVCS on a PC or VAX or both systems.

PVCS Maintains System Integrity

PVCS prevents corruption of code that could ordinarily result from security breaks, user carelessness or malfunctions. The levels of security can be tailored to meet the needs of your project.

PVCS & PolyMake Work Together

PolyMake, the leading MS-DOS make utility, is now available for the VMS operating system. This allows you to write makefiles that will function in both PC and VAX environments. Additionally, PolyMake reads time & date stamps of PVCS archives for fast, accurate program rebuilding.

PVCS and PolyMake Maintain Source Code Written In Any Language.

The Preferred Version Control System

The customers listed below are just a few of the innovative leaders that have made PVCS the leading version control program for personal computers.

Alcoa Aluminum
Arthur Anderson
AT&T
Ashton-Tate
Bank of America
Bell Labs
Bendix
Boeing
CIGNA
Citibank
3Com
Colonial Penn
Commerce Clearing House
Control Data Corp.
Corvus
CXI
Digital Equipment Corp.
Deloitte Haskins + Sells
Diebold
Dow
Dunn & Bradstreet
EDS
Educational Testing Service
E-Systems
Equitable Life
Federal Express
First Boston
Ford
Fox Software
Fujitsu
GTE
Hardees
Hewlett-Packard
Honeywell
Hughes Aircraft
IBM
Industrial Networking
Intel
ISC Aerospace
IVAC
Javelin
Lattice
Lawrence Livermore
Lotus
McData Corp.
McDonnell Douglas
Mead Data Central
MIT Lincoln Labs
Nastec
Novell
NCR Technologies
Pitney Bowes
Plexus Computers
Price Waterhouse
ROLM
Rockwell International
Safeco
Sears
Security Pacific
Sperry
Software Publishing
Spacelabs
Standard Oil
Standard & Poors
Tandem
Tektronix
Telex
Texas Instruments
Touche Ross
Unisys
United Airlines
United Parcel Service
United Technologies
U.S. West
Westinghouse Electronics
Xerox

Only PVCS meets the needs of independent programmers and corporations. Once you standardize on PVCS, the archives used to track and monitor changes are interchangeable between any PVCS product. You will receive full credit for your initial purchase if you upgrade to a higher-priced MS-DOS version of PVCS.

Personal PVCS — Offers most of the power and flexibility of Corporate PVCS, but excludes the features necessary for multiple-programmer projects.

Corporate PVCS — Offers additional features to maintain source code of very large and complex projects that may involve multiple programmers. Includes multi-level branching to effectively maintain code when programs evolve on multiple paths (e.g. new versions for different host systems, or a new program based on an existing program).

Network PVCS — Extends Corporate PVCS for use on Networks. File locking and security levels can be tailored for each project.

PVCS for VAX systems — Requires VMS. Uses the same interface and archive format as MS-DOS version. Supports branching and offers file locking and other security features for multiple-programmer projects.

	MS-DOS*	VMS		
	PC/XT/AT	Micro VAX II	VAX 7xx	VAX 8xxx
Personal PVCS	$149			
Corporate PVCS	$395			
Network PVCS	$995**	$4,950	$9,500	$10,500+
PolyMake	$149			
Network PolyMake	$447**	$1,250	$2,375	$2,500+

*Compatible with MS-DOS 2.0 through 3.3. Compatible with the IBM PC/XT/AT & other MS-DOS PCs.

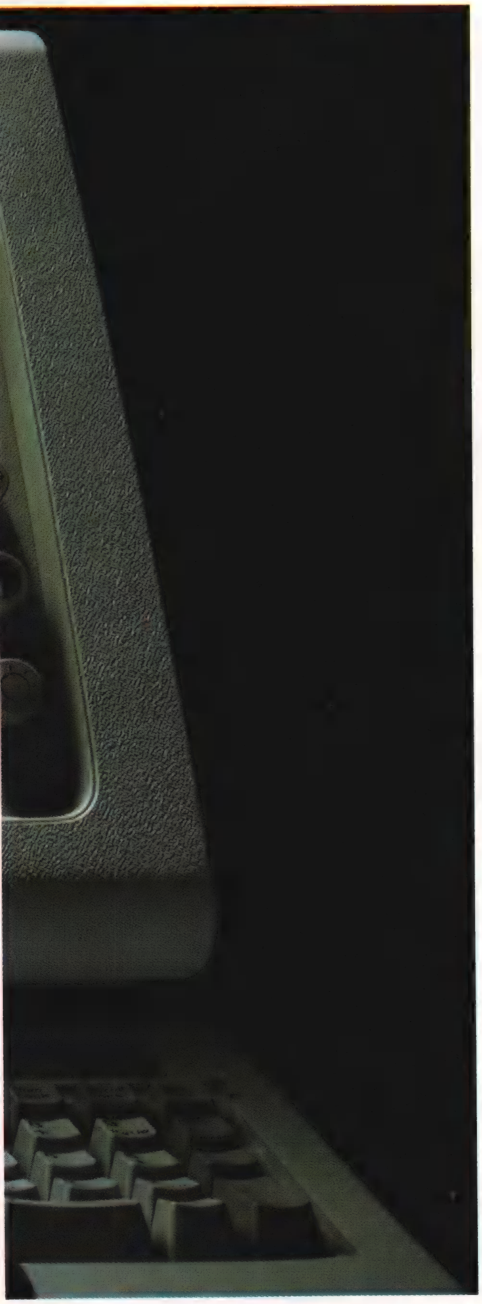
**5 Station LAN License. Call for pricing on larger Networks.

TO ORDER:
VISA/MC 1-800-547-4000.
Dept. No. 355
Oregon & Outside USA call (503) 645-1150.
Send Checks, P.O.s to: POLYTRON Corporation, 1815 NW 169th Place, Suite 2110, Beaverton, OR 97006.

POLYTRON

High Quality Software Since 1982

what serious expert Power tools.

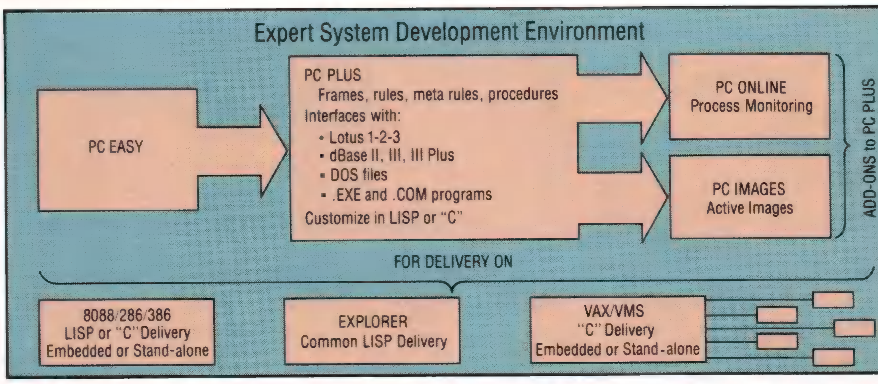


Among all the expert system development tools available for personal computers today, none deliver the power and flexibility of TI's Personal Consultant series.

Personal Consultant Easy is ideal for getting started, and is upwardly compatible with the higher functionality of PC Plus. For experienced developers, Personal Consultant Plus and its optional add-on enhancements, Online and Images, were designed to help solve a broader range of complex problems.

package helps deliver expertise that is "online all the time."

Application delivery as flexible as the tools themselves.
Delivery can be in LISP for flexibility, or "C"* for maximum speed and portability. Our "C" options support either stand-alone or "embedded" knowledge bases. Options are available for DOS-based PCs, TI's Explorer, and DEC's VAX™ line of multi-user minis running under VMS™.



Personal Consultant Plus. Full power for an affordable price.
At $2,950, PC Plus has proven to be one of the richest and most flexible problem-solving tools available for the development of complex knowledge-based systems. Designed to take advantage of today's more powerful 286/386 DOS-based computers, or TI's Explorer™ Symbolic Processing System, the new 3.0 version of PC Plus provides powerful standard features and a continuing growth path with the addition of either PC Images or PC Online, or both.

Personal Consultant Images. Picture an expert system with interactive graphics.
At $495, PC Images enables developers to create knowledge-based applications that incorporate complex graphical "active images." User-interactive dials, gauges, forms and selection images provide a more exciting visual data input and output style.

Personal Consultant Online. The expert system as part of the process.
At $995, PC Online allows the developer to design expert systems which interact directly with process data, as opposed to input from a human operator. Designed for intelligent process monitoring applications, this optional

"Texas Instruments has done more than any other company to educate people about AI, to popularize it, and to make useful AI tools available at reasonable prices."
— *Jim Seymour, PC Magazine.*

Technical support, training courses and Knowledge Engineering Services are available for the Personal Consultant products. If you have a question about any of our expert system power tools, we have the answer.

Pick up the phone and gain a powerful advantage.
Call **1-800-527-3500** for technical overviews of our products and a PC Plus case histories brochure which details how our power tools are being put to work today.

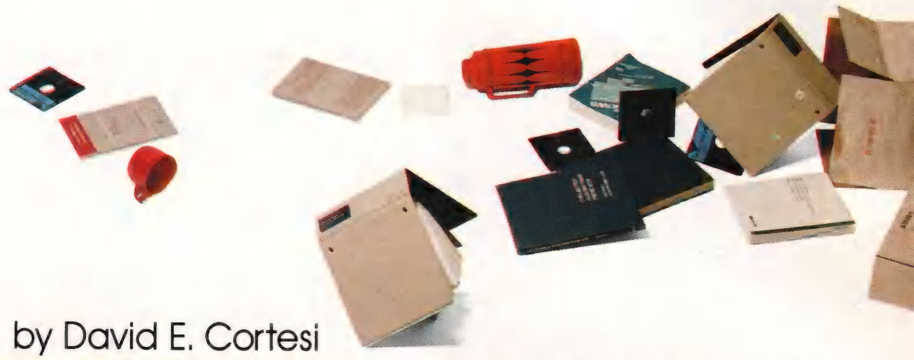
36106
© 1987 TI
Personal Consultant and Explorer are trademarks of Texas Instruments Incorporated.
dBase is a trademark of Ashton-Tate.
Lotus 1-2-3 is a trademark of Lotus Development Corp.
VAX and VMS are trademarks of Digital Equipment Corporation.
*Available 4Q 1987.

TEXAS INSTRUMENTS

knowledge bases
- Interactive dials, gauges, forms and selection images
- Multiple images can be combined on same screen
- "Getting Started" tutorial-style manual

Personal Consultant Online
- Optional add-on package for PC Plus (3.0)
- ONLINE expert systems that interact directly with process data
- Multiple interfaces to data acquisition and analysis programs
- Knowledge base synchronization with process data
- Functions for historical and predicted trends
- Special user interface/reporting capabilities
- "Getting Started" tutorial-style manual

Dynamic Linking in OS/2



by David E. Cortesi

Built-in facilities for the third-party extension of OS/2

No operating system is likely to include all the functions that creative programmers will demand. It appears that Microsoft, benefiting from past experience, has purposely designed OS/2 so that third-party developers can more easily add functions to it. Yet to maintain an acceptable degree of quality assurance, Microsoft wanted to make sure that any additions would fit seamlessly into the whole system without compromising its reliability. This article is a brief survey of the mechanisms included in OS/2 to allow this, and in particular, the concept of dynamic linking.

Briefly put, a dynamic link is an external reference that is not resolved at the time the program is linked. Instead, the connection to the external routine is made either while the program is being loaded or sometimes even later while it is executing.

Dynamic linking isn't a novel idea; it was fundamental to the operation of the Multics system (the Intel 80286 has some intriguing similarities to the GE 645, the Multics host) and readers of the DDJ Forum on CompuServe have told me that the Prime operating system PRIMOS and UCSD Pascal have similar features.

Preparing a Dynalink Library

Here's how dynamic linking works in OS/2. You design and build a package of code that will be useful to more than one program. At first you store its object code in an object library as usual and test it by linking it in the usual way with the programs that use it. When your package is in something like its final form, you run it alone through the linker in a special pass. You also supply a description file that tells the linker that this code will be a dynamic link library, or a dynalink, as it's come to be called. In the description file you specify the names of all the entry points that will be publicly available in this package. You may also specify attributes for individual segments; I'll come back to those later.

The linker processes the object code much as it does in MS-DOS, merging segments by class and group, resolving internal references between segments, adjusting offsets to account for merged segments. One step of linking in OS/2 is different from that with MS-DOS. In MS-DOS, the linker's output is a single, monolithic binary image in which the input segments have lost their individual identity. In OS/2, the linker keeps segments separate in the executable file. That's necessary so that the system loader can load a program one segment at a time, as it must do in order to build a local descriptor table (LDT) for the 80286 hardware.

The processed segments of your package of subroutines go into a file that has the same format as an OS/2 executable (.EXE) file, but by convention a dynalink library has the file type description. If your package is THEGOODS, its code will be linked as THEGOODS.DLL. That is all that needs to be done for code that will be bound to its caller very late, during execution time, but that's a rare use.

Most dynalink libraries need to be known to their client programs at link time. In that case, one more processing step has to be done. A utility called IMPLIB reads the description file and writes an artificial object library that can be used at link time. Applied to the

David E. Cortesi, 415 Cambridge St., #18, Palo Alto, CA 94306. Dave is a former DDJ columnist.

description file for your package, this produces a very small file named THEGOODS.LIB.

Using a Dynalink Library

Now client programs can begin using your package. They declare its entry points just as they would declare any other external references, whether in assembly language:

```
extern GoodyA:far
```

or in Pascal:

```
Procedure GoodyB(c:char);
   external;
```

or in C:

```
extern int far GoodyC( );
```

And the calls to the package's procedures are written just as they would be if the package were to be linked in the usual way (which it might still be in some cases).

When a client program is linked, the import library, THEGOODS.LIB, is one of the libraries input to the link. The object records in it are special. They don't contain object code; they only tell the linker the name of the dynalink library (THEGOODS.DLL, remember?) and the names of the entry points that it exports for use. The linker writes a table of these names into the linked program (let's say it's CLIENT.EXE) and pointers to where the references occur in the linked segments.

Load Time Linking

Eventually CLIENT.EXE will be loaded for execution. The linked segments of code and data will be brought in from the disk file—if necessary. It might not be necessary because it is possible to tell the linker to mark any segment "load on demand." In that case, the OS/2 loader won't load the segment but will only set up the local descriptor table to cause a hardware trap if the segment is referenced by an instruction. When and if that happens, the segment will be loaded.

Once it has storage copies of the program's segments, the loader processes dynamic links. The linker has given it the character string file name of the dynalink library and the names of the entry points needed. The loader looks up the file; it has to be found in a directory designated in the system configuration file.

The system may not have to load the segments of dynalink code. Dynamically linked code segments are shared among all clients that use them, so if CLIENT.EXE is the second instance of a program that uses THEGOODS, no disk input will be needed for code segments.

Because a dynalink library has precisely the same file format as a .EXE file, the process of loading dynalink code is really just an extension of loading a program. The only extra step is that the loader has to fix up the external references in the client code. Because the process is so similar, there's no reason why code in one dynalink library shouldn't call code in another one, and that in a third, and so on.

Benefits of Dynamic Linking

Some of the benefits of dynamic linking are clear at once. There's an economy of disk space: whereas under MS-DOS every client program contains a copy of the common code, OS/2 stores only a single copy in the dynalink library. There's economy of memory because only a single copy of common code is kept in storage. There's economy of load time because only the needed segments are brought in from disk. (What's faster than a disk cache? Not doing the disk input at all!)

(continued from page 19)

These features would be enough to justify extensive use of dynamic linking. And OS/2 does use it extensively. The whole interface to the operating system is based on it. *All* the OS/2 system functions are presented as external procedures that programs declare and call. And all of those system procedures are defined by code in dynalink libraries with names such as DOSCALLS.DLL, VIOCALLS.DLL, and MOUCALLS.DLL. As a result (and in sharp contrast to MS-DOS), it is actually easier to use OS/2 system calls from a high-level language than it is to use them from assembly language.

I promised this would be a discussion of system extensions; here's where the connection is made. Because the entire programming interface to OS/2 is through dynalinks—and because dynalinks may themselves call DynaLinks—any new dynalink library is a functional extension to OS/2 on an equal footing with the system code itself. There aren't any special interfaces, no magic incantations that only gurus may use; there isn't the sharp distinction between, say, ordinary programs and TSR programs that exists in MS-DOS. There is just the hierarchy of functions available in DynaLink libraries, with the OS/2 kernel dynalinks at the base of the pyramid.

And there is no lack of function, either. Don't suppose that, because a system extension is restricted to the same facilities that any program might use, it isn't possible to write interesting system extensions. There are a *few* limitations, and I'll mention them shortly. But consider this: all the functional extensions to be supplied in the IBM Extended Edition of OS/2 (database facility and multiple communications protocols) as well as the whole of the Microsoft Presentation Manager (protect-mode Windows), all of this code will be supplied as dynalink libraries. There won't be any features added to OS/2 at a kernel level to support them—they're all there in 1.0 and available (to high-level languages, remember) as dynalink calls. So if you repackage your B-tree access method as a dynalink library, it will use the same kernel functions and be exactly as accessible to its clients as IBM's database facility is to its.

Let's explore some of the subtleties of dynamic linking. Return to the point at which you are linking your package of code into a dynalink file. A "definition" file must be given to the linker to describe it. A definition file may be used when linking any program; that's how you tell the linker to mark a program segment for deferred loading. But there are two definitions that will most often be applied to DynaLink segments—one is shared data; the other is IOPL code.

Shared Data Segments

By default, data segments are not shared between programs. Instead, a new copy of a data segment is loaded from the .EXE or .DLL file for each instance of a program that uses it. That fits the expectations of most programs: you don't ordinarily think of a segment of data as being accessible to two or more concurrent programs at once.

You can tell the linker to mark a data segment "single," however—that is, that there is to be only one instance of that data segment in storage no matter how many client programs might have concurrent access to it. This simple concept forms the basis for some sophisticated applications.

Recall that, although such a data segment is common to all client programs, it isn't directly accessible to them. The data segment is linked with your code, not (directly) with the client program. If it doesn't contain any exported entry points, there is no way in which a client program can form an external reference to it. Though there are ways in which determined programmers could find out its segment address, there is no way they could find out how you arrange and manage its contents unless you tell them. The only convenient way that client code can access a dynalink data segment (shared or not) is by calling the procedures in the dynalink package.

Because a shared data segment is dynamically linked, it will be loaded with the first client program to request the package. It will remain in storage so long as at least one client program does. (If it's important to keep your dynalink package in storage at all times, you can write a dummy client program and start it with a statement in the configuration file.) The segment is an island of static data that your code can address easily (because it was linked with your code) but that *only* your code knows how to address. You can use it for a shared buffer pool, or for queues of work (however your package defines "work"), or generally for any kind of pooled resource that your package can usefully manage on behalf of multiple concurrent client programs.

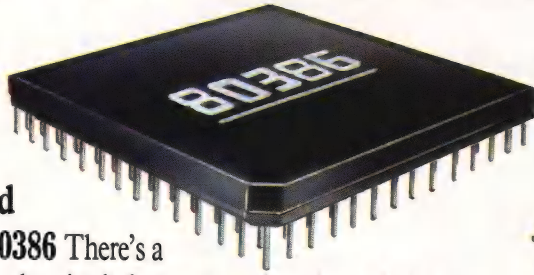
You can't predict how many clients may be executing in your package at any instant; there might be none or there might be dozens! But the OS/2 kernel has plenty of functions to help you manage shared data in a multiprogramming environment. There's a complete, and quite efficient, set of semaphore operators, so you can serialize access to the data. There's a generalized queueing facility so that multiple "writer" threads can queue data (using a variety of queue disciplines) for a single "reader" to process. There's a storage suballocation facility that has built-in serialization so multiple threads can allocate and free pooled storage concurrently.

I/O Privilege and Devices

In OS/2, application code runs at privilege level 3, in the 80286 scheme of things. The I/O privilege level is set at 2, so application code that tries to do an I/O instruction will trap out and be terminated. Any code segment, however, dynalink or not, can be marked at link time as being eligible for I/O privilege. Code in such segments may request access to a range of I/O port numbers from the kernel and then may do I/O instructions, including setting and clearing the interrupt flag.

There are serious restrictions on this I/O privilege, however. There is no way that application code can get control on an I/O interrupt. There is no way that application code can lock a segment in storage so as to use it for a DMA buffer. And because OS/2 is slicing time among potentially many programs, polling an I/O port is

A Number of Reasons A Number



1. Designed for the 80386

There's a revolution taking place in desktop computing. A revolution that's been launched by a square wafer of silicon known as the 80386 microprocessor chip. It puts minicomputer potential at PC users' fingertips. It's a fact that virtually every leading PC manufacturer has built a "box" around this chip. And it's a fact that the "New Operating System" will, supposedly, even run on it. But, it's also a fact that *their* system wasn't designed for the 80386. Ours is. And it's called PC-MOS/386.™

2. PC and PS/2 Compatible

In designing PC-MOS, we knew our first priority was to exploit the minicomputer capabilities of 80386-based PCs & PS/2s. But we went further, and developed a system which would be fully compatible with the millions of existing PCs, PC ATs, and PC-compatibles. Power without sacrifice. You'd expect nothing less from the new standard bearer.

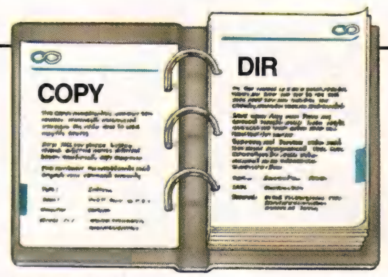
4. Thousands of DOS Programs

PC-MOS/386™ gives you the best of the past, and the best for your future. Which means that while PC-MOS/386™ totally replaces your old DOS, you won't have to replace the programs you've spent a lot of time learning.

And it all happens so effortlessly. You'll continue to reap the benefits of your favorite DOS programs, while entering a new arena of power.

Think of it! Programs like dBASE III, WordPerfect, Lotus 1-2-3 and Symphony, WordStar, MultiMate...literally thousands of DOS programs–all compatible and multi-user available.

5. Familiar Commands Like DIR and COPY



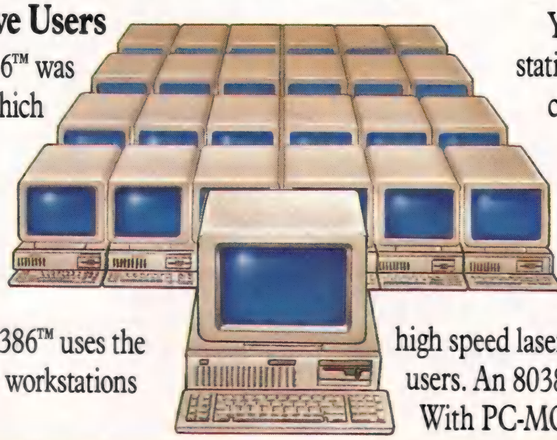
Just as you don't have to learn a whole new array of software to take advantage of PC-MOS/386,™ neither do you have to learn an entirely new set of commands.

Instead, the system builds on the knowledge you already have. "COPY" still copies files, and "DIR" still gives you a directory listing. As you might expect, we didn't stop there. There's a wealth of features that have strengthened the commands you know, making them more powerful and easier to use.

3. One, Five, Up to Twenty-five Users

From the beginning, PC-MOS/386™ was designed as a versatile operating system which could support twenty-five users as easily as it supports one. The system comes in single, five, and 25-user modules, so you're able to start with what you need and expand when you're ready.

In a multi-user setting, PC-MOS/386™ uses the computing power of the host PC to drive workstations linked to standard RS-232 ports.



You can choose from a variety of workstations. Mix and match dumb terminals costing under $500 each with PCs and PS/2s running our terminal emulation software.

All of the host's resources can be shared. Programs, data, hard disks, tape backup units & printers (including high speed laser printers) are suddenly available to all users. An 80386-PC has minicomputer potential. With PC-MOS/386™ you can "mini" your micro.

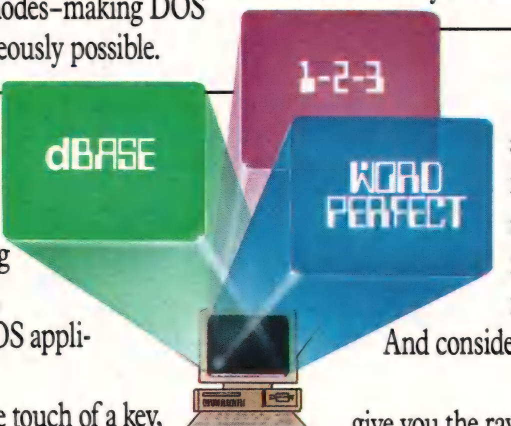
# of Users Will Choose PC-MOS/386.™

6. Concurrently Supports Virtual 8086 and 80386 32-Bit Mode

80386-based PCs & PS/2s are dual-personality computers. To run DOS programs, they act as PCs with a 640K memory limit. But to take advantage of their minicomputer capacity, they operate in true 80386 mode which lets them address up to four gigabytes of memory. PC-MOS enables the 80386-host and its workstations to independently switch between these modes–making DOS compatibility and 80386 power simultaneously possible.

7. Multi-Tasking

While it's true you could look elsewhere for multi-tasking, why would you want to? The *other* multi-tasking operating system is not now, nor is it planned to be, multi-user. It won't even run multiple DOS applications in multi-tasking mode.



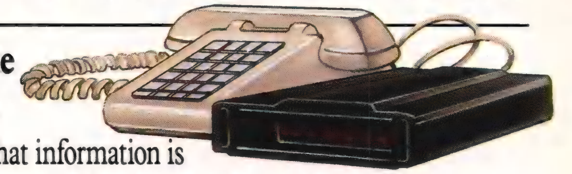
Now consider PC-MOS/386.™ At the touch of a key, you can switch between up to 25 different tasks. And if you have workstations connected to a host, they get multi-tasking, too. Finally...a system that won't hold you back.

8. File/Record Locking and Security

When you decide to implement either a network or a multi-user system, there's a two-fold problem which must be solved: protecting your work from accidental misuse and securing it from intentional theft.

PC-MOS/386™ solves both aspects of this problem. Password protected security allows you to assign file, directory, and task access to each user. Plus, files and records are locked using either PC-MOS' proprietary system or NETBIOS emulation.

9. Remote Access



It's been said that information is power...which makes PC-MOS/386™ a deadly weapon to your competition. Imagine on-the-road salespeople being able to file call reports and access your latest inventory data. Picture executives being able to access your corporate database from across the country, or around the world–giving them the information they need, when they need it.

Visualize branch offices tapping time-critical data with nothing more than a modem and a workstation. Working at a home office in the evening or over the weekend suddenly gets awfully productive. And that makes good business sense. The kind of sense you can't afford to be without.

10. The Price...

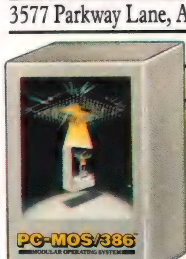
As you evaluate operating systems, ask yourself if it's reasons you're considering...or rhyme. Ask if you're getting a system for tomorrow, or one that was made for yesterday. See if you're being forced to buy new hardware because of *their* software.

And consider this.

Only one operating system in the world can give you the raw power, features, and functionality that you demand. Its name is PC-MOS/386.™ And it's immediately available in one, five and 25-user versions starting at $195.

For the dealer nearest you, **CALL: 800/451-LINK**	In Georgia: 404/441-2580	International/OEM Sales: 404/263-1006	Resellers/VARs: 404/448-5465

3577 Parkway Lane, Atlanta, GA 30092 Telex 4996147 SWLINK FAX 404/263-6474



The Software Link/Canada CALL: 800/387-0453

DEALER INQUIRIES INVITED

PC-MOS/386™

MODULAR OPERATING SYSTEM

THE SOFTWARE LINK

CIRCLE 110 ON READER SERVICE CARD

(continued from page 20)

impractical as well.

As a result, a package based on a specific piece of hardware will have to include a device driver. Device drivers have access to a set of kernel functions to assist them in translating between virtual and real addresses, in locking storage, in fielding interrupts, and so forth.

Unfortunately, OS/2 device drivers are even harder to write than MS-DOS device drivers. The reason is the uncertainty about the machine state at the time an interrupt occurs. The system might be operating in real-address mode in the simulated DOS 3.3 environment—or it might be operating in protect mode. The interrupt-handling code of a device driver has to be able to operate in either mode. Furthermore, the split between the "strategy" and "interrupt" halves of a device driver, a split that was only formal in MS-DOS, is real in OS/2. The strategy routine has to start, or queue, the work to be done and then get back to the kernel pronto, and the interrupt routine has to be able to operate asynchronously.

The expanded support for the *IOCtl* system call might make some device-centered packages easier to design. Application code can exchange data with device driver code through this call. (The distributed OS/2 device drivers support an elaborate scheme of "generic" *IOCtl* calls that bring a degree of device independence to this very low level of the system.) If I were designing a package based around a piece of hardware, I would try very hard to put as little function in the device driver as possible and reserve as much as possible to dynalinked code. I'd use the device driver to lock segments in storage and to field interrupts. Everything else could be done from dynalinked code that used the device driver as its private resource.

Dynalink Descriptors

Because I'm into technical details here, I might as well explore a strange addressing problem. This item is merely a sidetrack intended for those who know the 80286 hardware well; others can just skip ahead to the next section.

You might suppose that, as dynalink segments are shared across clients, they must be addressed through the Global Descriptor Table (GDT). Not so. OS/2 appears to reserve the GDT for kernel code and system data objects. Application code segments and dynamic link segments all go in the LDT of each task. It's also possible to share dynamically-allocated data segments between programs, and these shared segments also are addressed through the LDT, not the GDT as you might at first expect.

The reason is probably security. Dynalink code isn't meant to be a global resource; it's linked to a specific client program or programs. Shared segments are only for the use of the clique of programs sharing them. By putting their descriptors in the LDTs of the using programs, the system ensures that only the right programs can use them.

Those who know the 80286 really well will immediately spot a problem. Loaded code often contains *far* pointers to its own segments, and *far* pointers are constants embedded in the code. It follows that a dynalink segment must use the identical segment address in every LDT in which it is entered. If it didn't, its embedded *far* pointers would be correct for a call from some tasks but not from others.

In order to make that happen, OS/2 has to treat the 8,191 possible entry indexes into an LDT as a global pool—even though an LDT is a private resource of one program. When a dynalink library is first loaded, it is assigned a set of descriptor table entry numbers from the pool. It will use those segment numbers in whatever program's table it appears, and no other segment can use those numbers until this dynalink library loses its last client and is unloaded. There's clearly a potential for creating large, sparse LDTs. It remains to be seen if that will be a problem in production systems.

Errors and Exit Lists

Very well, a client program called your *InitGoody* entry point, then called your *StartGood* entry point to begin a complex operation. And now the client program has done something foolish and has been terminated by the system, leaving your shared data segment or I/O device in a halfway condition, storage allocated, semaphores uncleared. Other clients will suffer. That's not good; it's a principle of OS/2 design that one program's disasters shouldn't affect other programs.

For every program, the OS/2 kernel maintains an exit list—a list of code addresses that want to get control before the program is terminated. There's a system call to enroll an entry point in the exit list. You'd use it from the *InitGoody* entry point, enrolling your *GoodyByBy* procedure as an exit procedure. Now if the client makes a serious mistake, OS/2 will call your code and you can clean up the debris.

The reason it's a list, not just a single address, is that the OS/2 design anticipates that an application might be using several dynalink libraries and every one of them might want to register its own exit procedure. There's no promise about which one will be called first, but that shouldn't matter. Each one should only be concerned with resources that are in its private domain and none should even be aware of the others.

Device Monitors and Replacement Functions

The OS/2 named devices (*PRN*, *LPTn*, *COMn*, *SCREEN$*, and *KBD$*) can be opened as files and used via file handles, just as under MS-DOS. Or they can be addressed as device drivers using the generic *IOCtl* system calls. The keyboard, screen, and mouse are also supported by dozens of system calls with names such as *KbdCharIn*, *VioWriteCells*, and *MouGetPtr*. All these are directly available to dynalink code, as they are to applications.

There are yet two more levels of control over device I/O. Although available to any code, they're sufficiently complex that they'll probably only be used in dynalink packages. These are device monitors and replacement functions.

A device monitor is a piece of code that monitors the

(continued from page 24)

stream of data bytes that flow between a device and a program. The monitor receives a data packet with a call to the *DosMonRead* dynalink function and is expected to pass the packet on with a call to *DosMonWrite*. But there's no rule that it has to pass on every packet or that it pass on the same number of packets. A monitor is free to censor the data stream, to generate new packets, or to substitute packets.

Consider a monitor for the keyboard data stream. It sees all keystrokes, each neatly wrapped up in a "data packet" with flags for the current shift state and a millisecond time stamp. It can interpret the keystrokes as it wishes and substitute for them. In short, a keystroke monitor may be a keyboard "macro" program—and it doesn't have to field interrupts or be loaded in any special way.

A keyboard monitor is created by system calls. A program could set up its own keystroke monitor, but more likely one would be set up in dynalink code (and torn down in an orderly way in an exit procedure). Because it's easy to set one up, there can be more than one. OS/2 permits a whole pipeline of keystroke monitors to exist, each one getting the output of the last and providing input to the next, and they don't have to be aware of each other.

There is a separate logical keyboard for each of the 16 "screen groups," and a keystroke monitor sees only the strokes from the keyboard of one screen group (not necessarily the one its code is running in!). The printer devices, however, are global to all screen groups. A printer monitor sees all the data passing to its printer from all processes in the system. Data packets contain the process ID that produced them, and "open" and "close" packets are visible. This is the basic function needed to build a print spooler: a program that intercepts multiple data streams, saves the data on disk, and sends files on to the printer in orderly sequence. An extremely rudimentary spooler is built on this facility and distributed with OS/2, but there is plenty of scope for better ones to be built. There are intriguing possibilities for building completely transparent support for network printers and for building filters that translate printer control codes for one make of printer to another's. All of this happens, remember, at the level of application code, without any requirement for I/O privilege or hardware dependency.

The dynalink libraries distributed with OS/2 contain, as I mentioned, dozens of system calls for using the keyboard, the mouse, and the screen. The screen calls contain all that's necessary for a full-screen editor, for example.

The distributed code, however, assumes the presence of an IBM-compatible keyboard, an IBM-compatible screen adapter, and one of a small number of mouse devices. That isn't enough. The OEMs that sell OS/2 systems will have the know-how and development tools to build replacement device drivers and dynalink libraries that support their particular hardware under the standard calls. But there will be times when the code for a keyboard, mouse, or screen operation should be replaced at the level of an individual program or an individual screen group (multiple programs may run in a single screen group).

This need is provided for. It is possible for a program to "register" a replacement procedure for almost any of the supplied mouse, screen, or keyboard calls. From the moment of registration, whenever the supplied function is called, control will be transferred to the registered replacement. That procedure gets the same stacked parameters as the original function would have seen. It may choose to interpret them in a different way (permitting access to a wider screen, for example), or apply them to the hardware in a different way, or censor or expand on them in the manner of a device monitor, or perhaps just record them for performance monitoring. The Presentation Manager will probably use this facility to take over the mouse and screen calls in the screen group that it controls.

Run-Time Linking

I alluded to the ability to do a very late dynamic link, after the program has been loaded and while it is executing. There is a system call that will take the file name of a dynalink library and load it, returning a handle. There is another call that takes such a handle and the character name of an entry point and returns a *far* pointer to the entry point. The calling program may then call the dynalink entry point.

This isn't genuine linking because calls to the external code can't be freely embedded in the caller's code. Calls have to be indirect by way of pointers in storage, and the linkage must be established with the explicit system calls. Still, it's a facility with some important uses. Basically, it gives a program the ability to configure its own contents at run time. The components of a large subsystem may be linked independently, each as a dynalink library. The kernel of the subsystem may decide at execution time which of its components ought to be loaded. A communications subsystem, for example, might dynamically load its components for different line protocols in this way, as needed.

Coding Dynalinks

Between dynamic linking, exit lists, device monitors, and replacement device functions, it ought to be possible to extend OS/2 in about any direction. Some extensions are more obvious than others. For instance, a dynamic link library would appear to be about the ideal way to package a compiler's run-time library. Think what a reduction that would make in the size of compiled programs! Unfortunately, it looks as if the languages released with Version 1.0, at least, will have their run-time code in the old-fashioned kind of object library.

Something else obvious is that it would be very desirable to write dynalink modules in high-level languages. Here, alas, you run into a major glitch in the system design. The code generated by the IBM/Microsoft C, Pascal, and FORTRAN compilers is not suitable for dynamic linking!

The reasons aren't too hard to understand. A dynamically linked module can be entered from different pro-

(continued from page 26)

grams. It has to be prepared to use its caller's stack, and it cannot make any assumptions about the way its caller uses registers. Unfortunately, the code from most compilers *does* make assumptions about registers, lots of them. The IBM/Microsoft compilers are particularly dependent on two assumptions that just aren't true on entry to dynalink code: that *ss = ds* and that *ds* bases *DGROUP*.

Neither of these is a big problem. The assumption that the stack and data segments are identical was a lazy coding ploy (don't bother saving *ds*, just reload it by pushing *ss* and popping *ds*) that ought never to have been allowed. This assumption isn't true in large-model C anyway, so it shouldn't be too difficult to eradicate it from all the code generators and all the run-time libraries.

The assumption is that *ds = DGROUP* is more subtle. Every compiled module has a *DGROUP*, a segment of static data. When object modules are linked, the linker merges all *DGROUPs* into one segment and adjusts the offsets in all instructions as required. This works just fine for an application program because all its parts are linked in one run. Whatever object code it includes references the same *DGROUP*. But the code of a DynaLink library is linked in a separate run. The pieces of its *DGROUP* are formed then, and its *DGROUP* is nothing like the *DGROUP* formed when its clients are linked.

But a compiler's code generator may assume that there is only one *DGROUP* and that *ds* addresses it at all times after the initialization of the module. That ain't true when a dynalink procedure is entered; then *DS* addresses the caller's *DGROUP*. What's lacking is that the prologue to any public procedure should contain the sequence familiar to any assembly-language programmer:

```
push ds
mov ax, seg DGROUP
mov ds,ax
assume ds:DGROUP
```

And, of course, the exit code of any public procedure would have to contain *pop ds* to restore the register.

Current compilers do not generate this code, and as a result the code they produce can't be used as dynalink procedures! Not, at least, unless it has an assembly-language front end to switch *DGROUPs* for it.

One other irritating problem gets in the way of high-level dynalinks. Existing compilers are prone to generating automatic checks of one kind or another—for stack overflow, for subscript ranges, for whatever. Not all of these can be disabled. If even one of them is left, some kind of error-handling module will be included in the link. But the only way such modules have to handle an error is to issue a message to the console, and the only way they can seem to find to issue a console message is to use the compiler's file I/O mechanism, so the presence of even one stack check causes a major part of the compiler's run-time library to cascade into the dynalink code. And that often brings with it modules that won't link properly in a dynalink.

Microsoft has been thoroughly beaten up on for these problems at developers' conferences and claims to be working on solutions. It'd better be working hard. Dynamic link libraries are an extremely attractive facility of OS/2, and the first programming languages that support them properly will have a competitive edge in a system that (Bill Gates confidently says) will have ten million users by 1992.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 1.**

A RAM-Cache Manager in C

by Alan Deikman

Prevent disk bashing with RAM caching

To an application program in most operating systems, a read/write request appears to take place instantaneously—throughput is primarily dependent on the I/O speed of the devices being accessed. For some devices this is an acceptable limitation. After all, during the interval a program is waiting to receive user input from the keyboard, there usually isn't anything for the program to do but wait.

For random-access disk requests, however, the data being sought, which may not arrive for another 80 milliseconds or so, may actually have been in-hand only moments before. If a program had the intelligence and foresight to save that block of data in RAM, there might not be any need for disk access at all and, as such, no need for that 80-millisecond delay.

When accessing disk blocks, it's a common practice to access the same blocks over and over again and to access other blocks only once in a long while. The operating system portion of the disk is typically the most often accessed—in MS-DOS, this section holds the directories and the file allocation tables (FATs).

In a multiuser operating system, ideally, CPU time is diverted to work on some other task when a program requests a disk block, with control passing back to the primary program as soon as the required data becomes available. Overall system throughput is thus enhanced at the expense of the individual program. Even so, there's no advantage in letting an application program force the operating system to access the disk more than is necessary.

Alan Deikman, Software Services, Menlo Park, CA 94026-2106. Alan is a management, marketing, and computer consultant. He has been consulting since 1978 and has had an independent practice since 1985.

If such a program is used frequently, the capacity of the entire system can be greatly increased if each program is optimized for fewer time-intensive disk requests. For I/O-intensive applications, this becomes a critical issue. An accounting or database system does very little computation as a rule, and most of the time spent (after the user has entered input) is spent waiting to store or retrieve data.

Within this context, there are two primary options for increasing program performance:

1. Buy faster hardware.
2. Reduce the number of actual disk accesses required.

Disk caching is one of the least expensive solutions available to the programmer to optimize program performance.

Cache Theory

To make a disk cache, an area of RAM is set aside to hold the most recently used blocks of data. Each block is identified by block number. Whenever a request for a disk block read occurs, the cache area is checked first to see if the required block is available. If it is available, no disk access occurs, and the calling program uses the RAM copy of the block. That disk block is then tagged "most recently used" (MRU). When this happens it is called a cache hit.

If it turns out that a disk access is required (because a cache miss occurred), the block is obtained from the operating system as usual. Before the block is used, however, it's copied into the cache area, replacing the "least recently used" (LRU) block in the cache area, if any exists. It is then marked MRU.

When a disk block is written, it also is placed in the cache as the MRU block. In most applications it is desirable to write the block physically to the disk at that time, but sometimes it is better to wait until the block becomes the LRU block and is about to be overwritten. It is possible that the block needn't be written in the first place.

The routines in this article maintain a two-way linked list, called the LRU/MRU chain. The *cacallo()* routine sets up the original linked list to include all the blocks allocated, as illustrated in Figure 1, page 31. A –1 (0xFFFF) flags the end of the chain.

Whenever a block is designated the MRU, it is taken out of the linked list and reinserted at the end. This operation is performed by the function *cacnew()*. If this operation were to be performed on block 2 of the initial chain, the result would be as shown in Figure 2, page 31.

Cache Size and Application

There's a trade-off between cache size and net efficiency. If the cache is too small, the likelihood of hits is small. On the other hand, if the cache is too large, the CPU spends more time than necessary looking up disk blocks in the cache. (The MS-DOS manual warns against this, in the discussion on setting up the *BUFFERS* command in CONFIG.SYS; the *BUFFERS* command sets up a system cache of 128-byte records.)

Figure 3, below, shows the curve representing an application that uses caching with perfectly random disk accesses. The left side of the curve represents no caching, and for small caches a decrease in performance occurs. Performance increases to some theoretical optimum and then falls off. Diminishing returns ultimately make disk caching more of a burden than a benefit.

Most applications, however, are not truly random in the way in which they access the disk. There are cases in which disk caching is misapplied. Consider the situation in which a file is read from beginning to end. No block is ever read twice, so applying disk caching is merely adding overhead. If the file is one block larger than the cache area, and must be rewound and read again, then each read on the second pass will result in a cache miss. Thus it is never desirable to apply disk caching to sequentially read files of an unknown size.

In yet another case, when the cache memory is bigger than the disk file, disk caching becomes superfluous because it would have been better to read the whole file into RAM initially, then access the blocks directly, than to put up with the storage overhead of keeping MRU/LRU information.

Because there are so many variables associated with the efficiency of caches, I've provided the *cacstat( )* routine to obtain the statistics of the cache, allowing the user to adjust the size of the cache for optimum performance. The values returned are cache hits, cache misses, and cache adds. The sum of hits and misses represent the total number of times the cache was searched for a block. The optimum ratio of hits to total accesses depends on a number of variables such as the ratio of CPU/RAM speed to disk average access time.

Multiple Caches

One of the most glaring failings of having an operating system perform all the cache work for disk I/O is that the operating system isn't in a position to discriminate between different types of accesses. Also, most systems can't allocate cache memory on a dynamic basis. As a result, the operating system cache can be wiped clean of useful records just because the application made one pass of an input file, filling the cache with records that are

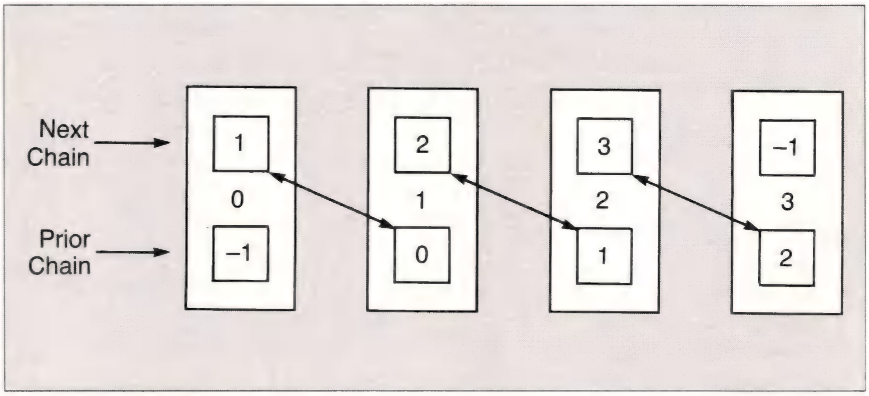


Figure 1: *Initial LRU/MRU chain*

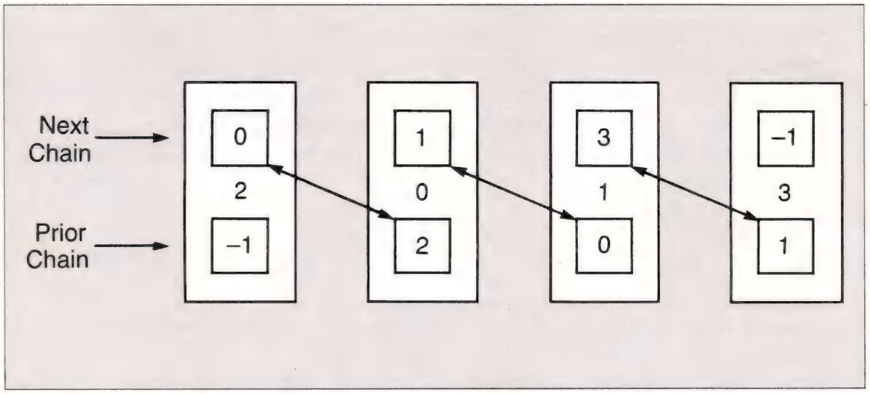


Figure 2: *After block 2 is made MRU*

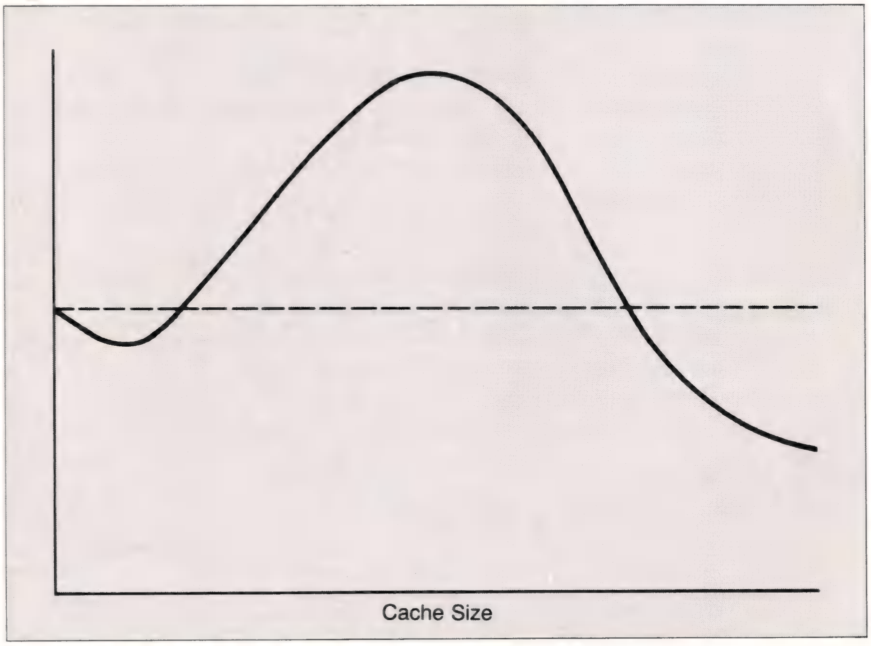


Figure 3: *Cache size vxs. performance*

RAM-CACHE MANAGER
(continued from page 31)

never read again. If the system is multitasking (or multiprocess), the situation would be even worse because other programs could dominate the available cache space.

It's usually obvious to the program designer which disk files should be cached and which shouldn't. A good application for a disk cache is a compiler's temporary tables, where using a disk scratch file is considered only after RAM memory runs out. In B-tree indexing subroutine libraries, caching is particularly effective in processing lookups and node additions.

For this reason, all the routines provided in this article operate on a structure that is pointed to by a single variable kept by the calling program. This approach allows any number of separate caches of variable sizes to be managed concurrently.

The *cacallo( )* routine uses the standard library *malloc( )* routine to allocate all memory necessary for the cache. It returns a pointer to the root structure of the cache that is used as a parameter to all the other cache routine calls. All the global parameters, and pointers to other objects, are contained in this structure, which is *typedef*ed to be *CACDS*.

Four parameters are required to set up a cache:

- the number of records in the cache
- the length of each record
- a pointer to an external function for processing free records
- an identifier word (*long*) to pass to the external function

If the application interface is not going to defer the writing of disk blocks, a free record-processing routine is not necessary. In this case, the third parameter provided should be a null pointer *(char *) 0*. The identifier word is used when a single routine is being used to handle the freed blocks from multiple caches. This value can be used by the called routine to identify from which cache the record is being transmitted.

Once again,
Compaq
raises the standard
of performance
for personal computers.

This time
by a factor of two...

Introducing the two on earth

The new COMPAQ DESKPRO 386/20™

Last year, we introduced the COMPAQ DESKPRO 386,™ the most advanced personal computer in the world. Now the world has two new benchmarks from the leader in high-performance personal computing. The new 20-MHz COMPAQ DESKPRO 386/20 and the 20-lb., 20-MHz COMPAQ PORTABLE 386 deliver system performance that can rival minicomputers.' Plus they introduce advanced capabilities, without obsoleting your investment in software, hardware and training.

Our new computers employ an industry-standard 20-MHz 80386 microprocessor and sophisticated 32-bit architecture. But to make these two of the world's fastest PC's, we did more than just increase the clock speed.

For instance, both are built around a concurrent bus architecture. Two buses—one for memory and one for peripherals—eliminate information bottlenecks, allowing each component

It simply works better.

most powerful PC's and off.

and the new 20-MHz COMPAQ PORTABLE 386™

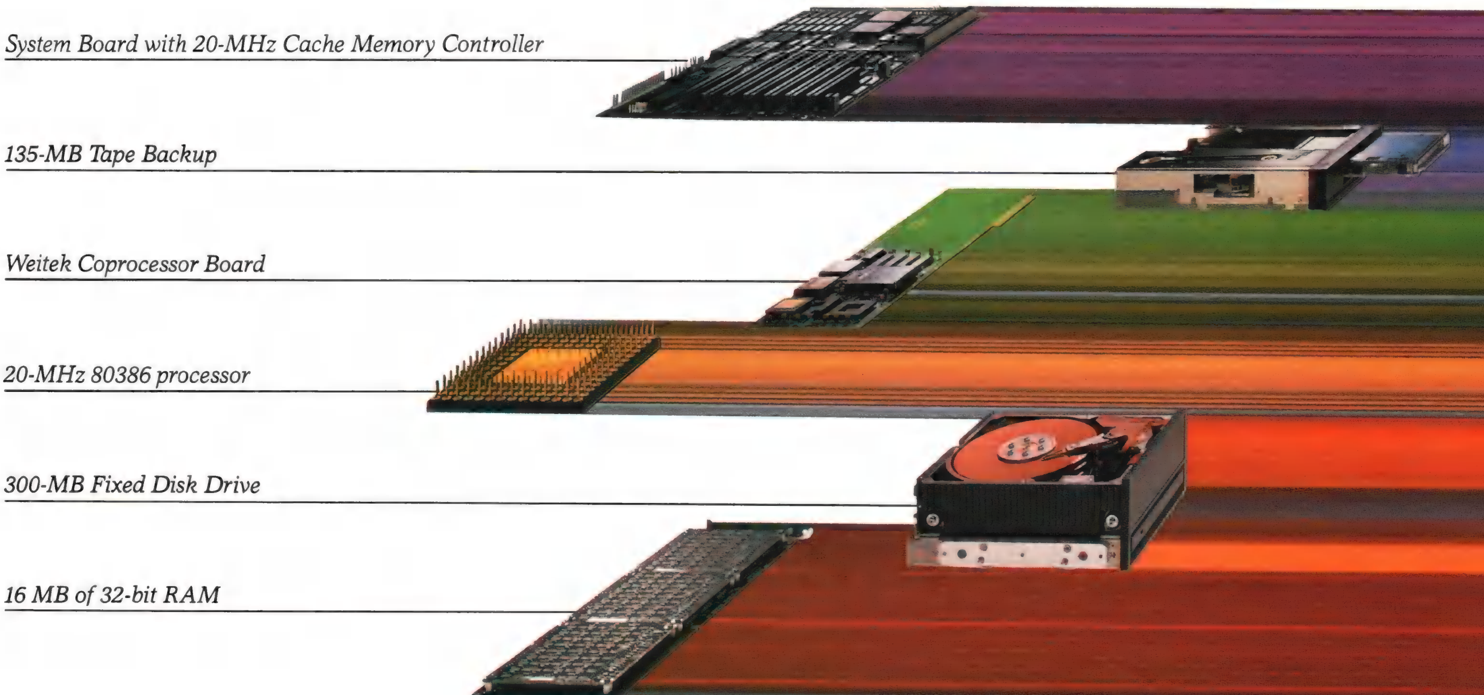
to run at its maximum speed. Together, they insure the highest system performance without sacrificing compatibility with industry-standard peripherals.

Both computers offer disk caching. Both offer the most memory and storage within their classes. Both let you run software being written to take advantage of 386 technology. And both run new MS-DOS®/BASIC Version 3.3 as published by Compaq. With it, our new portable and our new desktop can break the 32-megabyte limit on file sizes that handcuffs other PC's, allowing you to build files up to the size of your entire fixed disk drive.

And from now until December 31, 1987, both computers come with a free package of new Microsoft® Windows/386 Presentation Manager. It provides multitasking and switching capabilities with today's DOS applications to make you more productive. But that's just the beginning. To find out more, read on.

COMPAQ®

The question wasn't but how to get the



The most powerful personal computer in the world

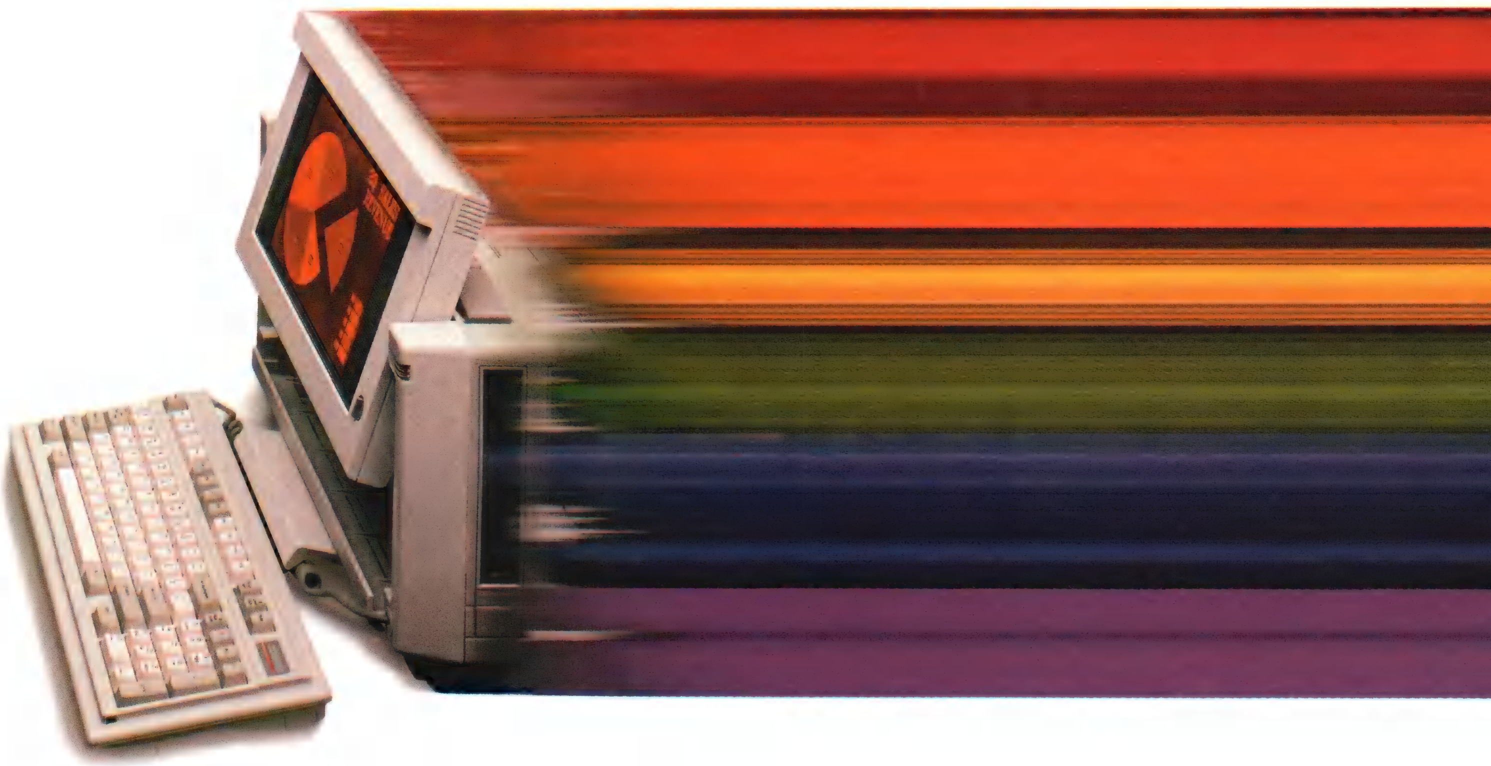
The COMPAQ DESKPRO 386/20 is an impressive 50% faster than 16-MHz 386-based personal computers. Even more impressive is the fact that it's up to 25% faster than other 20-MHz 386's. That's because the processor is just one small part of how the COMPAQ DESKPRO 386/20 outperforms every other PC in the world today and even many minicomputers.

The big reason is the new COMPAQ Flexible Advanced Systems Architecture, which optimizes overall system throughput while maintaining full compatibility with industry-standard peripherals. It does this by combining an advanced memory caching scheme with memory and peripheral buses that operate concurrently.

Complementing the speed of the microprocessor is the new advanced 20-MHz Intel® 82385 Cache Memory Controller. Like an efficient secretary that keeps frequently used information close at hand, it allows the microprocessor to operate at 0-wait states 95% of the time.

While one bus handles these high-speed operations, another *simultaneously* handles periph-

Everyone expected Compaq But no one



Pound for pound, it is the world's most powerful computer

Compaq has long been recognized as the world leader in both 80386 technology and portable computing. So it isn't surprising that we would combine the two.

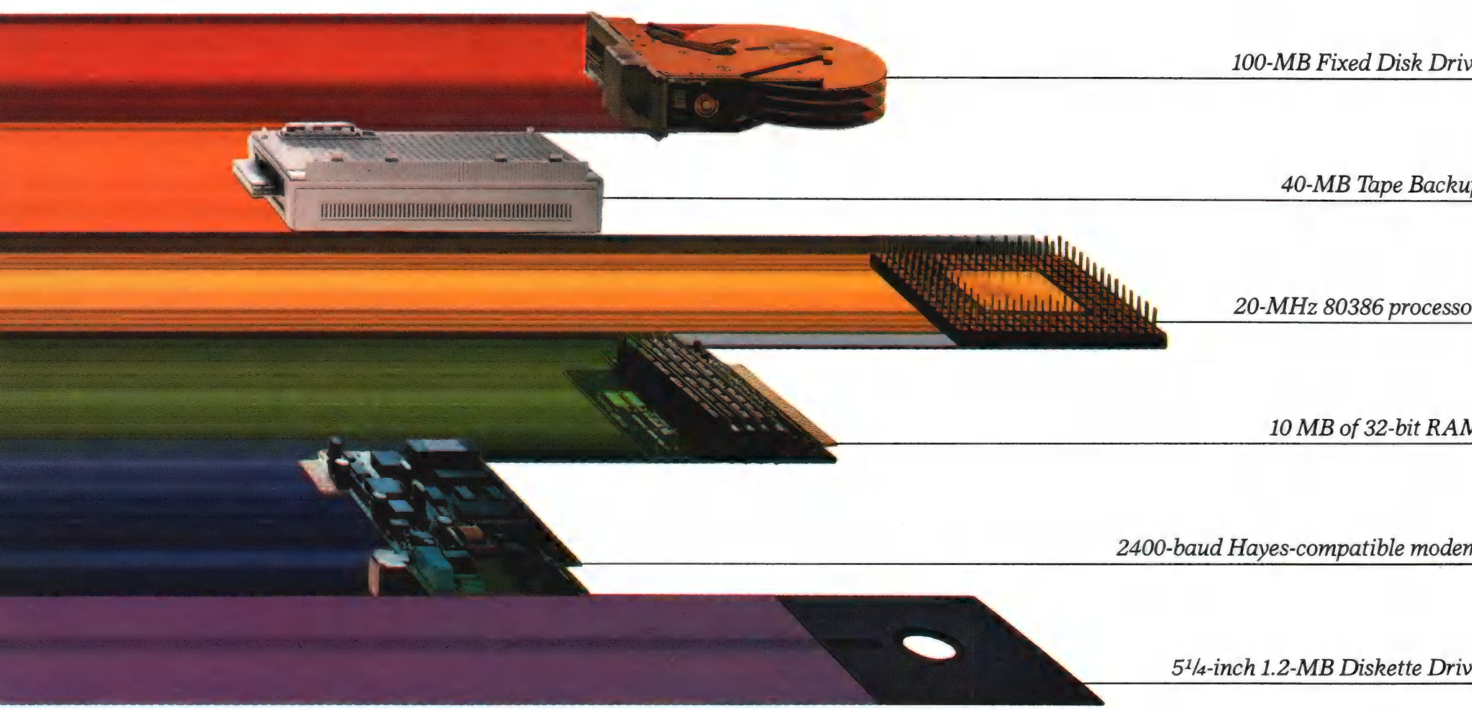
But no one expected the new COMPAQ PORTABLE 386 to run at 20 MHz. And no one even dreamed that it would offer 100 megabytes of storage, disk caching, and much, much more.

Our newest 20-lb. portable computer goes far beyond an 80386 microprocessor with a handle. It's not just the most advanced portable in the world. Pound for pound, it's the world's most powerful computer. Period.

Like the recent COMPAQ PORTABLE III,™ which changed the shape of full-function portable computing, the COMPAQ PORTABLE 386 makes no compromises. It offers more speed, memory, storage and features than any other portable PC. It runs your current software up to 25% faster than 16-MHz 386 PC's. Beyond that, its performance in calculation-intensive

It simply works better.

to introduce a 386 portable PC. expected all this.



applications is increased even more when you add an optional 20-MHz 80387 coprocessor.

Memory? Get one megabyte of 32-bit, high-speed RAM standard or go as high as 10 MB internally. And like all of the COMPAQ 386-based PC's, it features the COMPAQ Expanded Memory Manager.

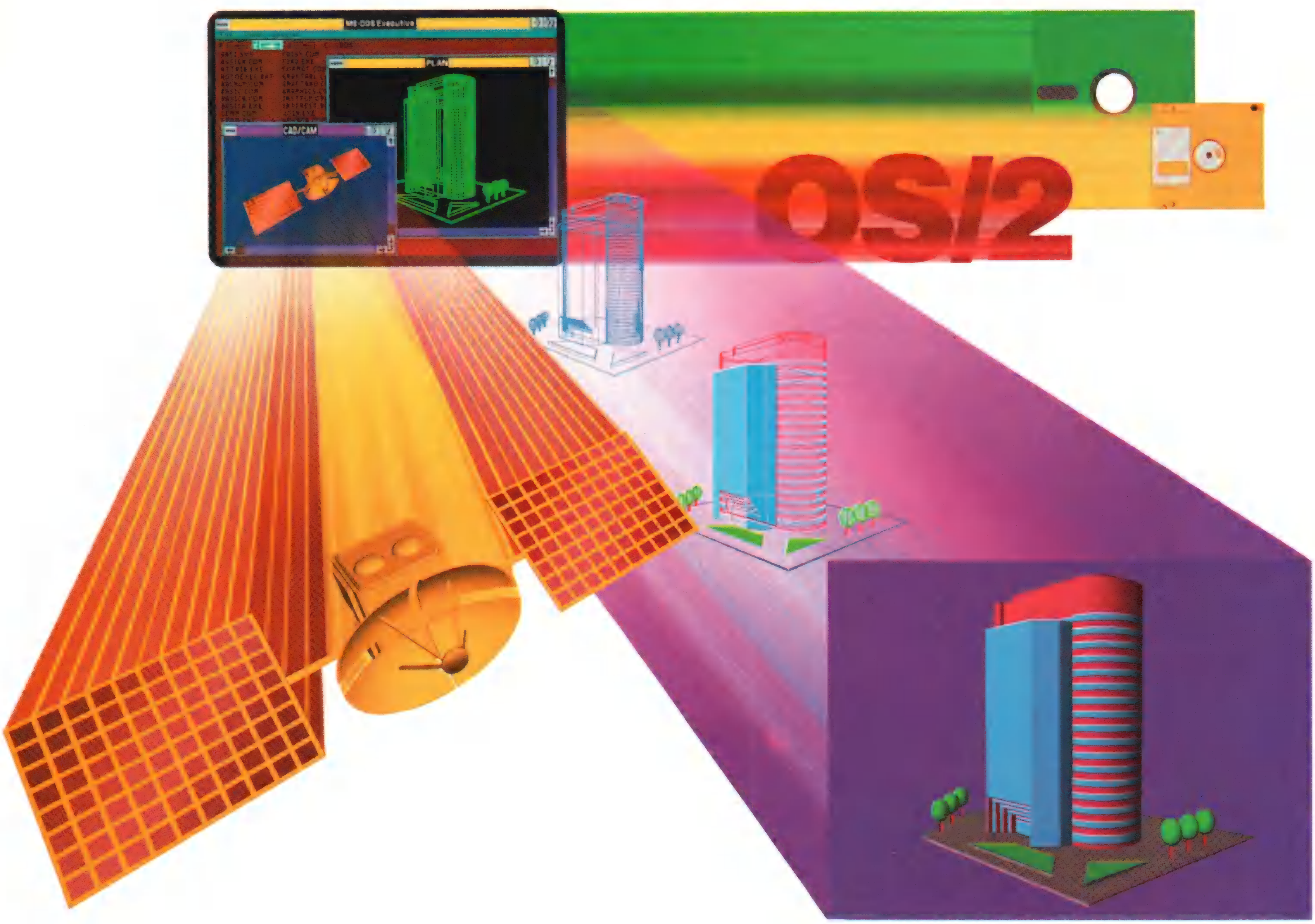
With our high-performance 100-megabyte internal fixed disk drive, you can actually fit 500 lbs. of data-filled pages into a 20-lb. PC, unsurpassed storage for a portable. If that's too much for you, we also offer a 40-megabyte model.

We've become famous for building desktop computer capabilities into our portables without leaving anything out. The COMPAQ PORTABLE 386 is more proof. It has a high-resolution, 640 × 400, 10-inch plasma display; a full-size, portable enhanced keyboard; two industry-standard expansion slots in a lightweight, optional plug-on unit; a choice between an optional 2400- or 1200-baud Hayes®-compatible modem; a full-size 5¼-inch 1.2-MB diskette drive; even an optional 40-MB tape backup.

These features, combined with the ultimate in portable performance, make the COMPAQ PORTABLE 386 the *biggest* PC this *small*.

COMPAQ
PORTABLE 386™

Compaq moves you ahead without leaving you behind.



Compaq offers the most complete line of high-performance 386 solutions. They all run industry-standard software and hardware, protecting the investments you've already made.

At the same time you won't be left behind when other technologies become important. Multi-task with existing applications using Microsoft Windows/386 Presentation Manager. Add VGA graphics if you wish. Run OS/2 when it's available. And now 3½-inch drives are even an option for our desktops.

We optimize the most advanced technology while maintaining compatibility with the past, present and future. This makes COMPAQ PC's a wise decision for serious business users. Because at Compaq, we don't burn bridges, we build them.

See the COMPAQ DESKPRO 386/20 and COMPAQ PORTABLE 386 at an Authorized COMPAQ Computer Dealer. *And from now through December 31, 1987, get Microsoft Windows/386 Presentation Manager free when you buy a 386-based COMPAQ computer.* For more information, call 1-800-231-0900, Operator 40. In Canada, call 416-733-7876, Operator 40.

Weitek™, Lotus®, Intel®, Microsoft®, MS-DOS®, Hayes® and OS/2™ are trademarks of their respective companies.

COMPAQ®

It simply works better.

Using the Cache Routines

All the routines and the *typedef* for the cache structure are in the file cache.h, shown in Listing One, page 62. It's not really necessary for the calling routine to have access to *typedef* because the calling routines never need direct access to the *CACDS*.

Cache.h declares the structure in *typedef*, however, so that the calling routine can conform to ANSI specifications by declaring an external or local variable of type *CACDS*. (With most compilers it is acceptable for the calling routine to store the returned value of *cacallo( )* in a character pointer.)

The cache.h file also provides the function declarations necessary for function type checking by the compiler. Listing Two, page 62, shows all the cache-processing routines, which may be compiled separately and/or incorporated into a subroutine library. Listing Three, page 67, is a simple test program for these routines.

After the cache has been set up, it will have to be checked to see if the desired record is already stored there. To do this, the *cacfind( )* routine is called. If the cache record designated by the *num* parameter isn't found, a null pointer is returned and the calling routine can then take the appropriate action. Usually this means issuing a *read( )* request to get the desired block, then adding the record to the cache according to the procedure that follows.

If the cache record is found by *cacfind( )*, a character pointer to the desired record is returned to the calling routine. That record is also taken out of its current position in the LRU chain and added at the MRU end. For example, suppose the calling routine is accessing blocks of 512-byte records and wants to read the record whose number is stored in the variable *recn*. The external character pointer *cache* is initialized with the return value of an earlier call to *cacallo( )*. The code (sans any error checking) might appear as shown in Example 1, right.

This example doesn't add any records that were read from the disk to the cache. A record produced by the *read( )* call should be added to the cache at the MRU end because, after all, it's the most recently used record. Each time a record has to be added to the cache, the *cacnum( )* routine is called. It is important to note that the *cacnum( )* routine does not check to see if the added record is already in the cache. Thus each call to *cacnum( )* should be preceded by a *cacfind( )*

```
char buffer[512];
long recn;
char *rec;

{
if ((rec = cacfind(cache, recn)) == NULL) {

      /* record not in cache, must be read from disk */

      lseek(fd, recn * 512, 0);
      read(fd, buffer, 512);
      rec = buffer;
      }

/* rec points to record to process */
}
```

Example 1: *Accessing blocks of 512-byte records*

```
char buffer[512];
long recn;  /* record number to write */
char *rec;  /* pointer to block in cache area */

{
if ((rec = cacfind(cache, recn)) == NULL) rec = cacnum(cache, recn);
memcpy(rec, buffer, 512);     /* copy data into cache area */
lseek(fd, 512 * recn, 0);
write(fd, rec, 512);
}
```

Example 2: *Writing blocks of 512-byte records*

(continued from page 33)

call.

The *cacnum()* routine does the following:

1. Finds the LRU record in the cache area.
2. If that record has been marked for processing, calls the external free block-processing routine.
3. Makes that record the MRU.
4. Numbers that record with a new number supplied by the calling routine.
5. Returns a character pointer to that record.

```
char buffer[512];
long recn;  /* record number to write */
char *rec;  /* pointer to block in cache area */

{
if ((rec = cacfind(cache, recn)) == NULL) rec = cacnum(cache, recn);
memcpy(rec, buffer, 512);            /* copy data into cache area */
cacproc(cache, recn);             /* mark the block for processing */
}
```

Example 3: *Code to defer writing records*

```
write_cache(idnt, recn, recb)
long   idnt;          /* cache identifier */
long   recn;          /* record number */
char   *recb;         /* record buffer */
{
   lseek(fd, recn * 512, 0);
   write(fd, recb, 512);
   return;
}
```

Example 4: *Example function to write a record*

To avoid having to copy blocks from place to place in RAM, and to eliminate the need to allocate an external buffer, the calling routines should access the records through a character pointer returned by *cacfind()* and *cacnum()*.

To complete the previous example for reading records, the following code (again without error checking for simplicity) can be used:

```
long recn;
char *rec;

{
if ((rec = cacfind(cache, recn)) = =
                                   NULL) {

   rec = cacnum(cache, recn);
   lseek(fd, recn * 512, 0);
   read(fd, rec, 512);

   }

/* rec points to record to process */
}
```

Writing data records undergoes a similar process. A record that is written also becomes the MRU record via the *cacfind()* and the *cacnum()* routines. If the record is already in the cache, however, there is no need to call *cacnum()*. The complementary routine to the previous one might look like that shown in Example 2, page 33.

Processing Freed Records

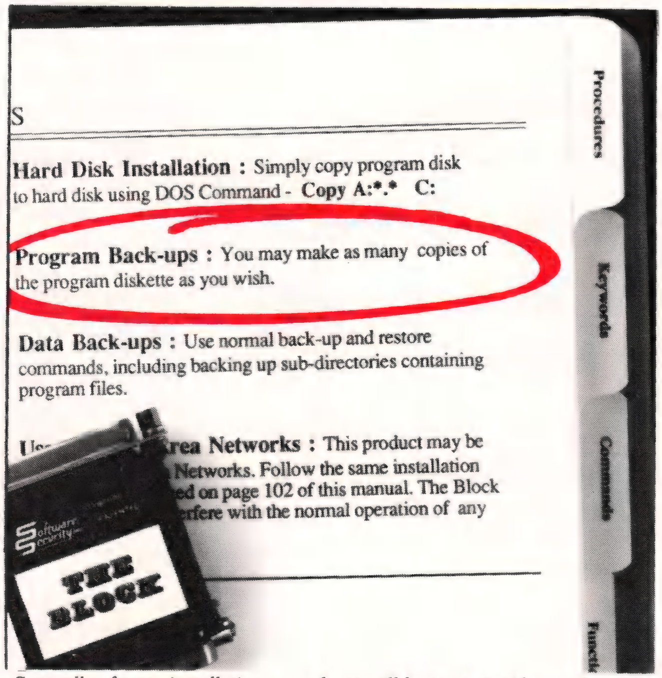
As mentioned earlier, it is sometimes advantageous to enter outgoing records (those to be written to disk) into the cache without actually performing the disk I/O operation, then write them to disk only when the block within the cache that the record is sitting on needs to be used for some other purpose. Typically, this would be the case for a scratch file (such as a symbol table), where the file would not even be created unless the data overflowed the cache. In most other cases, it is best to write the data records immediately, as in Example 2.

If the writing of data records is to be deferred, the third and fourth parameters to the *cacallo()* function are used and the *cacproc()* function is used to mark records that are to be processed before the space they

"How to protect your software by letting people copy it."

By Dick Erett, President of Software Security

Inventor and entrepreneur, Dick Erett, explains his company's view on the protection of intellectual property.



Soon all software installation procedures will be as straightforward as this. The only difference will be whether you include the option to steal your product or not.

"A crucial point that even sophisticated software development companies and the trade press seem to be missing or ignoring is this:

Software protection must be understood to be a distinctively different concept from that commonly referred to as copy protection.

Fundamentally, software protection involves devising a method that prevents unauthorized use of a program, without restricting a legitimate user from making any number of additional copies or preventing program operation via hard disk or LANs.

Logic dictates that magnetic media can no more protect itself from misuse than a padlock can lock itself.

Software protection must reside outside the actual storage media. The technique can then be made as tamper proof as deemed necessary. If one is clever enough, patent law can be brought to bear on the method.

Software protection is at a crossroads and the choices are clear. You can give product away to a segment of the market, or take a stand against the theft of your intellectual property.

"...giving your software away is fine..."

We strongly believe that giving your software away is fine, if you make the decision to do so. However, if the public's sense of ethics is determining company policy, then you are no longer in control.

We have patented a device that protects your software while allowing unlimited archival copies and uninhibited use of hard disks and LANs. The name of this product is The BLOCK™.

The BLOCK is the only patented method we know of to protect your investment. It answers all the complaints of reasonable people concerning software protection.

In reality, the only people who could object are those who would like the option of stealing your company's product.

"...eliminating the rationale for copy-busting..."

Since The BLOCK allows a user to make unlimited archival copies the rationale for copy-busting programs is eliminated.

The BLOCK is fully protected by federal patent law rather than the less effective copyright statutes. The law clearly prohibits the production of work-alike devices to replace The BLOCK.

The BLOCK attaches to any communications port of virtually any microcomputer. It comes with a unique customer product number programmed into the circuit.

The BLOCK is transparent to any device attached to the port. Once it is in place users are essentially unaware of its presence. The BLOCK may be daisy-chained to provide security for more than one software package.

Each software developer devises their own procedure for accessing The BLOCK to confirm a legitimate user. If it is not present, then the program can take appropriate action.

"...possibilities... limited only by your imagination..."

The elegance of The BLOCK lies in its simplicity. Once you understand the principle of The BLOCK, hundreds of possibilities will manifest themselves, limited only by your imagination.

Your efforts, investments and intellectual property belong to you, and you have an obligation to protect them. Let us help you safeguard what's rightfully yours. Call today for our brochure, or a demo unit."

Software Security inc.

870 High Ridge Road Stamford, Connecticut 06905
203 329 8870

CIRCLE 119 ON READER SERVICE CARD

RAM-CACHE MANAGER
(continued from page 34)

occupy is to be overwritten. The function specified by the calling routine is called with three parameters:

1. the cache identifier (the fourth parameter to *cacallo( )*)
2. the record identifier (generally the block number)
3. a character pointer to the record to process

To implement this, the previous record-writing routine would be changed to the code in Example 3, page 34.

The calling routine must provide an external function to write the record, specified by the initial call to *cacallo( )*. When a block marked for processing is needed for some other purpose, the routine is called. An example is shown in Example 4, page 34.

In some applications, a block marked for processing will become obsolete. In this case, the routine *cacuprc( )* is called. This routine will keep a block from being sent to the external function that writes the records.

Closing Down the Cache

Because *cacallo( )* uses *malloc( )* to allocate all the memory the cache uses, it's possible to free that memory with *cacfree( )*. All the records that are due for postprocessing are processed at this time by using the *cacflsh( )* routine, before the library routine *free( )* is called.

Variable Length Records

The cache routines presented in this article are best suited to small- and medium-size caches of large, fixed-length records. Typically, the "records" cached are disk blocks, which contain smaller, logical records. When the records are small, the linear search would become inefficient in short order. Figure 4, right, shows a block diagram of a complete application in which interface programs process requests from the applications.

Conclusion

The routines provided herein can be applied to almost any application. They were originally implemented on a Unix System III, 68000-based system and were subsequently ported to other environments. They work well under MS-DOS using all the memory models of the Microsoft C compiler, although a cache of more than 63K does not work.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listings begin on page 62.)

Vote for your favorite feature/article.
Circle Reader Service **No. 2.**

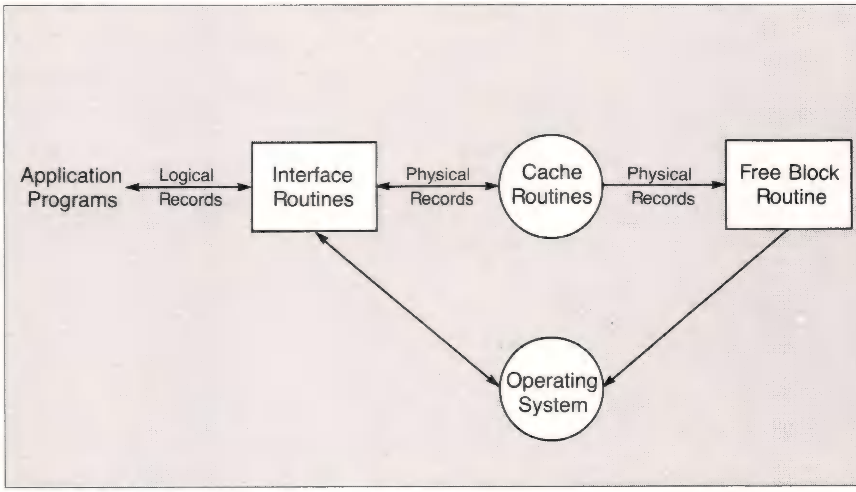


Figure 4: *Application interface*

Putting ROM Code in Its Place

by Rick Naro

A DOS Locate utility

Until recently, developers programming for EPROM-based applications didn't have a utility that would take the standard output of MS-DOS tools and spit out a file suitable for burning into EPROM. After spending the last few months longing to use the latest in compilers, such as Microsoft's C or Borland's Turbo C, I decided that perhaps the best tack to take would be to develop my own locate utility.

Although it's an easy matter to put the contents of a .COM file into an EPROM, programs in .EXE format are a different animal altogether. Unlike the binary image of a program found in a .COM format, a .EXE file is a relocatable object module, which requires that the segment references in a program be relocated or adjusted.

Before the .EXE file can be run, a loader must convert the relocatable format into an executable, absolute object module. In the typical MS-DOS system, program loading and segment fix-up is performed by COMMAND.COM and as such is transparent to the user. The COMMAND.COM loader simply reads the relocatable object module into memory, performs the segment fix-up by adjusting the segment references relative to the base segment of the load module, then transfers control to the program.

To support relocation, the .EXE file is partitioned into two components—a header containing the relocation information and the actual binary load module. Both the memory requirements for the program and the initial register values are found in the header, so the first step of loading the binary image is easy.

COMMAND.COM simply requests a suitably sized memory block in which to place the load module, then reads the binary image into that block of memory. Once loaded into memory, segment references are fixed relative to the base segment, using the segment fix-up records. When a programmer codes the following instruction sequence, for example, the assembler and linker cannot determine the final segment value to be used for the *data* segment, and the fix-up is left for the loader to perform:

```
mov ax, data ; Load ax with the data
                                segment
mov ds, ax ; Store in ds
```

Instead, the linker inserts the offset (or virtual segment) of the segment *data* from the base of the load module into the binary object module and inserts an entry in the segment fix-up record pointing to the segment reference requiring fix-up. If the program base is segment *3000h* and the virtual segment of *data* is *1234h*, the loader will perform the fix-up by adding the two and overwriting the segment offset with the sum *4234h* (which is the physical segment for the segment *data* in this instance). Once all segment fix-ups have been processed, the loader can transfer control to the new program.

An MS-DOS Locator

For embedded systems, a special type of loader called a locator is required. A loader is distinguished from a locator in two ways: by output destination and by the organization of the absolute object module. Although a loader is designed to write the absolute object module directly to memory (for immediate execution), the output of a locator is an Intel extended hex file suitable for EPROM burning.

Another important feature of a locator is its ability to rearrange segments at arbitrary addresses to reflect the physical organization of the target system. A typical embedded system normally contains EPROM at the upper addresses for the program code and RAM at the lower addresses for data, interrupt vectors, and the stack. Because this organization is incompatible with the contiguous MS-DOS absolute object module, relocating the segments to new addresses is crucial to the operation of the locator. When the locator has finished processing a .EXE file, ROMable code and data will be fixed at addresses in the EPROM address space and volatile data and the stack will be fixed in the RAM address space and the segment fix-ups adjusted to reflect the rearrangement of segments.

Rick Naro is a part time programmer who is interested in software development tools for embedded systems. He can be reached c/o Paradigm Systems, P.O. Box 152, Milford, MA 01757 or as naro on BIX.

By itself, the .EXE file header contains insufficient information for relocation, so the segment map of the program along with instructions on where the segments will be placed in the target system is required. The segment map is prepared by the linker and identifies each segment by name, class, length, and its position within the binary load module. The user must also prepare a configuration file describing the characteristics of the target system and the physical addresses that the program segments will bound. Using both the map and configuration files, the locator can extract and physically relocate the segments to build the ROMable load module.

Although a programmer is normally concerned with segments, they are far too numerous and varied in name to be of much use to the locator. Instead, the locator works with classes. A class is simply a tag applied to a segment by the assembler or compiler. This tag permits the linker to group a segment with other related segments.

For example, each separately compiled source file in the large-memory model will generate a uniquely named code segment, but all such segments will belong to the *code* class. Using the locator directives, a programmer can fix the address of any class and specify the order of a set of classes, configuring the absolute object code for any target hardware.

The locator also needs to process the segment fix-ups but in a slightly different manner from the way the loader does. Each segment listed in the segment map is given an entry in a linked list that contains its segment name, length, virtual segment number, and physical segment number organized by class name. When the configuration file is read, the base segment used for a class is fixed and all physical segment numbers are adjusted by adding this value to the segment offset within the class. Segment fix-up then can proceed with the virtual segment number from the fix-up record used to scan the linked list looking for a match. If found, the corresponding physical segment number is returned and used in the fix-up; otherwise, an unresolved segment error is reported.

Although the location process sounds simple, there are two pitfalls that must be avoided. As noted previously, an MS-DOS .EXE file is designed to be executed from a contiguous block of memory whereas embedded systems typically have a fragmented address space with pockets of RAM and EPROM placed at the whim of the hardware designer. A potential problem exists in that two adjacent segments in different classes can share a common virtual segment and then be located noncontiguously when the segments are extracted. Because the virtual-to-physical-segment translation in this instance is ambiguous, a situation known as segment aliasing results. Segment aliasing can be avoided by guaranteeing that two segments in different classes never share a common virtual segment. This is easily accomplished by verifying that the first segment in a class is paragraph-aligned or that each segment spans a paragraph boundary.

There is also a potential problem is using groups. A group is a collection of unrelated segments that are organized to fit within, and be addressed as, a single physical segment. Some linkers, such as the Microsoft linker, don't include sufficient information for the locator to reconstruct a group. If groups are used, the user must have information on the organization of the group and include instructions in the configuration file to permit its reconstruction. This is accomplished by using the locate directives to fix the address of the first class in the group and then order the remaining classes in the group as specified by the compiler vendor.

LOCATE

LOCATE is an MS-DOS utility that accepts a relocatable .EXE file and outputs an absolute object module suitable for burning in EPROM. The source code and a make file for building LOCATE can be found in the accompanying listings (beginning on page 68).

Because each application is unique, LOCATE uses several directives to control the location process. These directives are used to identify ROMable classes, assign physical segments to classes, and specify the order of classes in the absolute load module. Some directives accept a list of one or more operands. The *[* and *]* characters are used whenever an operand is optional and can be repeated zero or more times. Unless otherwise specified, directives and operands are delimited by white space.

The default configuration file has the file name of the input .EXE file with an extension of .CFG. Using a command-line option, the default file name can be overridden and any file can be specified to contain the configuration instructions. This option allows multiple load modules to share a common configuration file.

Class Directive

The *class* directive assigns a physical segment to a class. The first segment in the specified class is assigned the base segment number and the remaining segments in the class are assigned segments relative to the first segment in the class. These segments depend on the length of the preceding segments and the segment alignment.

The *class* directive uses the following syntax:

```
class class = seg
```

where *class* is the name of the class and *seg* is the 16-bit physical segment where the class will be located. For example:

```
class code = 0xfc00
```

assigns the class *code* to segment *fc00h* and therefore the physical address *fc000h*.

Order Directive

The *order* directive is used to specify the ordering of two or more classes. It is important because it allows unrelated classes to be made contiguous without firsthand knowledge of the size and number of segments in the class.

The *order* directive uses the following syntax:

```
order class [class]
```

where the first class in the list was

47% SAVINGS

Yes! I want to subscribe to Micro/Systems Journal™ and save over 47% off the cover price.

☐ 1 Year (12 issues) $29.97 ☐ 2 Years (24 issues) $56.97

Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment Enclosed ☐ Bill me later

Card # ____ Exp. date ____

Signature ____

Name ____

Address ____

City ____ State ____ Zip ____

Savings based on a full one-year cover price of $47.40. Canada and Mexico add $6 for surface mail, $14 for airmail per year. All countries add $24 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

2003

A PUBLICATION OF M&T PUBLISHING, INC.

47% SAVINGS

Yes! I want to subscribe to Micro/Systems Journal™ and save over 47% off the cover price.

☐ 1 Year (12 issues) $29.97 ☐ 2 Years (24 issues) $56.97

Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment Enclosed ☐ Bill me later

Card # ____ Exp. date ____

Signature ____

Name ____

Address ____

City ____ State ____ Zip ____

Savings based on a full one-year cover price of $47.40. Canada and Mexico add $6 for surface mail, $14 for airmail per year. All countries add $24 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

2003

A PUBLICATION OF M&T PUBLISHING, INC.

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Micro/Systems Journal

Box 3713
Escondido, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Micro/Systems Journal

Box 3713
Escondidio, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Micro/Systems Journal

Box 3713
Escondidio, CA 92025-9843

(continued from page 39)

specified in a *class* directive. Any class names specified after the first are located contiguously and aligned to the segment boundary of the first segment in each class. For example:

```
order code data bss
```

orders the classes *data* and *bss* immediately following the class *code*.

Dup Directive

The *dup* directive is used to make a copy of the specified class. If used, the *dup* directive should appear before any other directives.

The *dup* directive uses the syntax:

```
dup class dup_class
```

where *class* is an existing class and *dup_class* is the name given to the copy of *class*. For example, the directive:

```
dup data const
```

makes a copy of the *data* class named *const*. This command is used in conjunction with the *order* directive to locate the *data* and *bss* classes in RAM but force a copy of the class *data* to be included in EPROM for power-on initialization of any initialized data.

If the class *data* contains the initialized data from a compiler, the following commands will locate *data* at address *1000h* and create a copy of *data* called *const* to be placed after the *code* class. The start-up code can then initialize the class *data* by copying the *const* class to the *data* class.

```
dup data const ; Copy the class and
                      call it const
...
class data = 0x100 ; Fix data at
                      address 01000h
class code = 0xfc00 ; Fix code at
                      address fc000h
...
order code const ; Const to immedi-
                      ately follow code
           ; And read by the startup code
```

Rom Directive

The *Rom* directive is used to specify

DOS LOCATE UTILITY
(continued from page 41)

which classes are ROMable. Classes containing program code and constant data need to be located and placed in ROM to be available when the system is powered up and initialized. Other classes such as uninitialized data and the stack require only to be located at a physical address and do not need to be placed in the output file.

The *rom* directive uses the follow-

```
  /*   This program demonstrates the use of the LOCATE utility. It
       contains all of the components of a typical C program to
       exercise the startup code and locate utility.

*/
char     *ptr = "class DATA" ;            /* Initialized data */
int array[10][10]  ;                      /* Uninitialized data */

main()
{
    int i,   j ;                          /* Automatics */
    static  char    *s = "" ;             /* Static initialized data */

    for (i = 0; i < 10; i++)    {
        for (j = 0; j < 10; j++)
            array[i][j] = i * j ;
    }

    strcpy(s, ptr)  ;           /* Bring in a library function */)
```

Example 1: *The demonstration program*

```
C>masm /MX tc, tc, tc ;
Microsoft (R) Macro Assembler  Version 4.00
Copyright (C) Microsoft Corp 1981, 1983, 1984, 1985.
All rights reserved.
  49272 Bytes symbol space free
     0 Warning Errors
     0 Severe  Errors

C>tcc -c -ml demo
Turbo C  Version 1.0  Copyright (c) 1987 Borland International
demo.c:

       Available memory 293342
```

Example 2: *Compiling the C source code and the MASM start-up module*

```
C>type demo.map

 Start   Stop    Length  Name             Class

 00000H  00035H  00036H  _TEXT            CODE
 00036H  0007FH  0004AH  DEMO_TEXT        CODE
 00080H  000A8H  00029H  STRCPY_TEXT      CODE
 000B0H  000BFH  00010H  _ETEXT           CODEEND
 000C0H  000D3H  00014H  _DATA            DATA
 000E0H  001A7H  000C8H  _BSS             BSS
 001A8H  001A8H  00000H  _BSSEND          BSSEND
 001B0H  003AFH  00200H  _STACK           STACK

 Address           Publics by Value

 0000:0000         START
 0003:0006         _MAIN
 0008:0000         _STRCPY
 000B:0010         TEND
 000C:0000         _PTR
 000C:0000         IDATA
 000C:0020         ARRAY
 000C:0020         BDATA
 000C:00E8         EDATA
 001B:0200         TOS

Program entry point at 0000:0000
```

Example 3: *The linker map file*

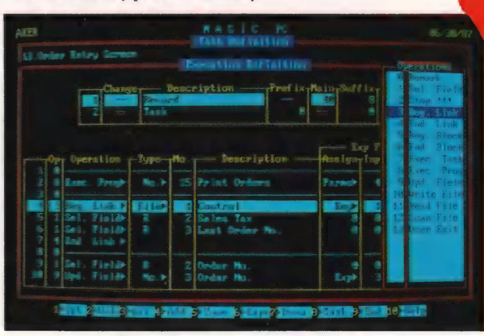
MAGIC PC: A REVOLUTION IN POWER, PRICE & PROGRAMMING SPEED.

You know how database applications are created — by hacking out line after line of time-consuming code. Most DBMS' and 4GL's give you some programming power. But when it comes to serious applications, they keep you bolted to your seat writing mountains of tedious code. And rewriting it all over again with every design change.

Imagine how much faster you'd be if you could replace the painful coding phase with an innovative visual technology which takes only a fraction of the time: Introducing Magic PC—the revolutionary Visual Database Language from Aker Corporation:

High-Speed Programming:

With Magic PC's visual design language you quickly describe your programs in non-procedural Execution Tables. They contain compact programming operations which are executed by Magic PC's runtime engine. You fill-in the tables using a visual interface driven by windows and point-and-shoot menus. One table with 50 operations eliminates writing more than 500 traditional lines of code. Yet with Magic PC you don't sacrifice any power or flexibility.



With a powerful set of high-level non-procedural operations you program at only a fraction of the time.

Maximum Power AND Simplicity:

With Magic PC, you can generate robust DBMS applications including screens, windows, menus, reports, forms, import/export, and much more! Plus, Magic PC has one of the friendliest user interfaces you've ever seen. Using Magic PC you can look-up and transfer data through a powerful Zoom Window system. Magic PC even lets you perform command-free queries.

Btrieve Performance:

Magic PC incorporates Btrieve, the high-performance file manager from SoftCraft. This gives you exceptional access speed, extended data dictionary capabilities, and automatic file recovery!

Virtually Maintenance-Free:

With Magic PC you can modify your application design "on the fly" without any manual maintenance. Magic PC automatically updates your programs and data files on-line! This also makes Magic PC an ideal tool for prototyping complete applications in hours instead of days.

FREE Networking:

Magic PC comes complete with LAN features. Develop multi-user applications for your LAN with Magic's file and record-locking security levels.

Stand-Alone Runtime:

Distribute your applications and protect your design with Magic PC's low cost runtime engine.

All For Only $199:

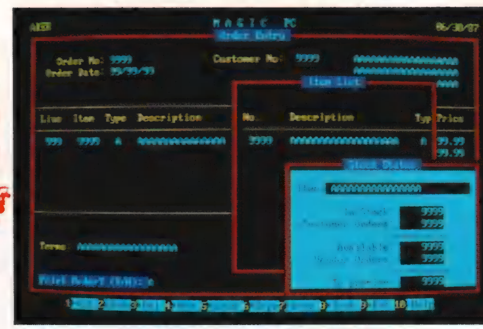
Best of all, Magic PC is an unbeatable bargain. For a limited time, Magic PC's price has been reduced to only $199! Yes, this is the same Magic PC that normally lists for $695! And Magic PC eliminates the need for a separate DBMS, compiler, or application generator. It comes complete with all the tools you need to develop your own database applications instantly.

CIRCLE 124 ON READER SERVICE CARD

$199 - With A Money-Back Guarantee!

For a limited time, you can get Magic PC for only $199. And even at this low price, Magic PC is risk-free. If you're not completely satisfied, simply return it within 30 days and we'll buy it back (less $19.95 restocking fee). And if you'd like a preview, Magic PC's Tutorial Demo is available for just $19.95.

But you'd better hurry – Magic PC's special $199 price won't last long!



Pop-up Zoom Windows run multiple programs per screen — with point-and-shoot data transfer between windows!

Join The Magic PC Revolution

To unleash your DBMS design power, order your $199 copy of Magic PC right now by calling toll-free or returning the coupon below.

ORDER NOW: CALL
(800) 345-MAGIC
In CA (714) 250-1718

"Magic PC's data base engine delivers powerful applications in a fraction of the time... there is truly no competitive product."
Victor Wright – PC Tech Journal

Also recommended by: *PC Magazine, PC World, PC Week, Computer Language, Data Base Advisor,* and many other publications worldwide.

MAGIC PC
The Visual Database Language
by AKER

Yes! I want to generate powerful applications much faster!

☐ Rush me my copy of Magic PC at the special promotional price of $199 (add $10 P&H, and tax in CA. International orders add $30). I understand I can return Magic PC for a refund within 30 days, if I'm not completely satisfied.*

☐ Rush me a copy of Magic PC Tutorial Demo at $19.95 (add $5 P&H, and tax in CA. International orders add $15).

Name ______
Company ______
Street Address (no POB) ______
City ______ State ______ Zip ______
☐ Check enclosed   Charge to my: ☐ VISA ☐ MasterCard ☐ AMERICAN EXPRESS
Account No. ______
Acct. Name ______ Exp. Date ___/___
Signature ______
Return to: Aker Corp., 18007 Skypark Cir B2, Irvine, CA 92714

DD

System requirements: IBM PC, XT, AT, PS/2 or 100% compatible with 512K RAM, hard disk and DOS 2.0 or later. 5¼" format, not copy protected. Dealer pricing available. *Return policy valid in US only.

DOS LOCATE UTILITY

(continued from page 42)

ing syntax:

rom class [class]

For example:

rom code const

forces the classes *code* and *const* to be placed in the output object file.

Comments

To aid in documenting the location process, comments can be added to the tail of any command line or as a separate line. Comments begin with a semicolon (;) and continue to the end of the line. Blank lines and comments can appear freely within the configuration file for documentation and readability.

Options

In order to provide a degree of flexibility, the LOCATE utility can accept command-line options (or switches if you prefer) that influence the operation of the locator. Command-line options are lowercase letters introduced with a leading dash (–) with no white space between the option letter and the argument. Some examples of LOCATE command lines are:

locate –b hello
locate –b –ccommon.cfg hello
locate –hhello.hx –b hello

A command line begins with *locate*, is followed by zero or more options, and is terminated with the path name of the file to be located. In the following descriptions, left and right brackets ([and]) are used to denote mandatory arguments.

–b—The default setting for LOCATE is to generate an Intel extended hex start address record containing the entry point of the program. By specifying the *–b* option, LOCATE will create an absolute segment at address *ffff:0* and place an intersegment jump instruction to the entry point of the program.

–c[filename]—specifies a different configuration file. The default configuration file is *filename .CFG*, where *filename* is the name of the load module. One use of this option is to allow different object modules to be located using a shared configuration file.

–h[filename]—changes the name of the Intel extended hex output file. Normally, the output is placed in a file with the same name as the .EXE input file and a default extension of .HEX.

–p[filename]—changes the name of the locate map file containing the segment assignments and public symbols. Normally, the locate statistics are placed in a file with the same name as the .EXE input file and a default extension of .LOC.

LOCATE Example

To demonstrate the use of LOCATE, I'll now discuss an example that uses the Turbo C compiler from Borland. Using the large-memory model, the program in Example 1, page 42, loads several different code and data segments that can then be processed by LOCATE. The com-

```
;
;       This configuration file is used with Turbo C to build a
;       ROMable image. It defines physical addresses for three
;       classes, makes a copy of the initialized data class to
;       keep in ROM and instructs the locator the order of the
;       different classes.

dup     DATA CONST            ; Make a copy of the initialized data class
                              ; and name it CONST

class  CODE = 0xfc00          ; Start code at address FC000H
class  STACK = 0x0080         ; The stack at address 00800H
class  DATA = 0x0100          ; DGROUP at address 01000H
order  DATA BSS BSSEND        ; Define the order of DGROUP
order CODE CODEEND CONST      ; And the order of classes in ROM

rom CODE CONST                ; ROM only the program code and the copy
                              ; of the initialized data that the startup
                              ; code copies from ROM to DGROUP
```

Example 4: *The configuration file*

DOS LOCATE UTILITY
(continued from page 45)

piled C source and Turbo C run-time routines are linked with the Turbo C assembly-language start-up code, TC.ASM. The start-up code together with the *order* and *dup* directives demonstrates how the locator can initialize the *data* class and zero out the *bss* class.

As shown in Example 2, page 42, you begin by compiling the C source and the MASM start-up module. The Turbo C options used are to disable linking (*–c*) and select the large-memory model.

I disable the automatic link following the compile so that I can substitute a ROMable version of the C start-up. The Turbo C large-memory-model library is searched to satisfy the external reference to the *strcpy( )* function, as follows:

```
C>tlink /m tc demo, demo, demo,
                         \turboc\lib\cl
Turbo Link Version 1.0 Copyright (c)
          1987 Borland International
```

For reference, the linker map file for this example is reproduced in Example 3, page 42. Note the segment and class assignments and watch how the locator processes and converts the executable image to a ROMable image.

The configuration file for the example (see Example 4, page 45) must be able to handle the group and the initialized data generated by Turbo C. LOCATE is instructed to make a copy of the *data* class that contains the program initialized data. Next the base segments of the three independent classes (*code*, *data*, and *stack*) are specified using the *class* directive. The *order* directive is used to recreate the Turbo C *dgroup* and fix the copy of the initialized data segment immediately following the *codeend* class. With it tucked nicely in EPROM and its physical address determined by the *tend* label, the start-up code can copy the class *const* to the *data* class before calling *main( )*.

LOCATE is then executed to process the .EXE file and output the absolute load module, as follows:

```
C>locate –b demo
MS-DOS Locate Utility
Copyright (C) 1987 Paradigm Sys-
          tems. All rights reserved
```

The output from the locator, shown in Example 5, page 46, is an Intel extended hex file and a segment map detailing the new physical address assignments. The locate map also contains a list of public symbols for use in debugging the target system. Note how the segments and classes have been relocated according to the instructions in the configuration file and correspond to the addresses of the target hardware.

The file DEMO.HEX (in Example 6, page 46) is now ready to be sent to the EPROM programmer. In an 8-bit system, the data is burned directly into one or more EPROMs. In a 16-bit bus system, the EPROM programmer must be used to split the load module into upper and lower bytes for programming the upper and lower bytes in separate EPROMs.

Summary

Although a simple example, the sample program demonstrates the power and flexibility of turning a low-cost PC into a powerful, embedded, system development tool for the NEC and Intel microprocessors. With access to a wide range of popular software development tools, program development for embedded systems has never been easier.

DDJ

(Listings begin on page 68.)

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**

```
C>type demo.loc
MS-DOS Locate Utility Version 1.0

Input File: DEMO.EXE
Output File: DEMO.HEX
Configuration File: DEMO.CFG
Invoked by: C:\BIN\LOCATE.EXE -b demo
Date/Time: Mon Aug 03 14:40:17 1987

Segment Information
Name          Class          Address     Length_
TEXT          CODE           FC000H      0036H
DEMO_TEXT     CODE           FC036H      004AH
STRCPY_TEX    CODE           FC080H      0029H
_ETEXT        CODEEND        FC0B0H      0010H
_DATA         DATA           01000H      0014H
_BSS          BSS            01020H      00C8H
_BSSEND       BSSEND         010E8H      0000H
_STACK        STACK          00800H      0200H
_DATA         NEWDATA        FC0C0H      0014H
??BOOT        (ABSOLUTE)     FFFF0H      0005H

Public Symbols
FC00:0000   START                  FC03:0006   _MAIN
FC08:0000   _STRCPY                FC0B:0010   TEND
0100:00E8   EDATA                  0100:0020   BDATA
0100:0020   _ARRAY                 0100:0000   IDATA
0100:0000   _PTR                   0080:0200   TOS

Entry Point - FC00:0000
```

Example 5: *The output from the locator*

```
:02000002FC0000
:10000000FAB880008ED0BC0002B800018EC0B80BD8
:10001000FC408ED8BE00008BFEB920002BCFF3A48D
:1000200061F32C0BF2000B9E8002BCFF3AAFB9A0D
:06003000060003FCEBCA10
:02000002FC03FD
:10000600565733F6EB2433FFEB1A8BC6F7E7508BC4
:10001600C6BA1400F7E28BD88BC7D1E003D858894B
:1000260087200004783FF0A7CE14683FE0A7CD7FFD0
:10003600360200FF360000FF360600FF3604009A3F
:0A004600000008FC83C4085F5ECBD5
:02000002FC08F8
:100000005657558BECFC1EC47E0E8BF732C0B9FFE1
:10001000FFF2AEF7D18CC38EDBC47E0AF3A41F8B34
:09002000560C8B460A5D5F5ECBB5
:02000002FC0CF4
:10000000080000113000001436C61737320444138
:040010005441000057
:02000002FFFFFE
:05000000EA000000FC15
:04000003FC000000FD
:00000001FF
```

Example 6: *The file DEMO.HEX*

Clarify and document your source listing and get an "organization chart" of your program's structure

with two NEW utilities from Aldebaran Laboratories, for C, BASIC, Pascal, dBASE,® FORTRAN and Modula-2 programmers.

Now works with FORTRAN

"Occasionally, a utility comes along that makes a programmer's life much easier. SOURCE PRINT is such a program. It contributes to the programmer's job by organizing code into a legible format and by helping to organize the documentation and debugging process."

— PC Magazine Sept. 16, 1986

Source Print and Tree Diagrammer both have easy-to-use menus with point-and-shoot file selection, and let you search for files containing a given string. For IBM PC and compatibles with 256K.

Join thousands of programmers who are working more efficiently using Source Print and Tree Diagrammer. Order these indispensable tools today. We ship immediately, and there's no risk with our 60-day money-back guarantee. **Order both and save. Only $155.00.**

800-257-5773 Dept. 58
In California:
800-257-5774 Dept. 58

MasterCard, VISA, American Express, COD. Add $5 for shipping/handling.

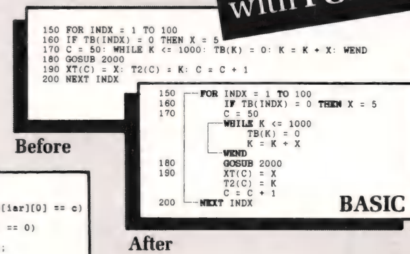
or see your local dealer!

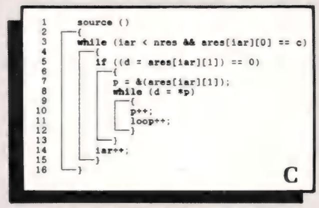
Source Print and Tree Diagrammer are trademarks of Aldebaran Labs. dBASE is a trademark of Ashton Tate. Prices subject to change without notice.

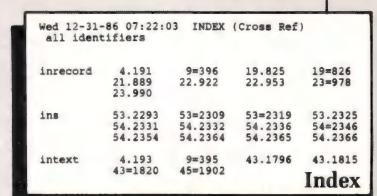
Source Print™

organizes your source code, simplifies debugging, and makes documentation a snap! It lists one or more source files with informative page headings and optional line numbers, while offering invaluable features:

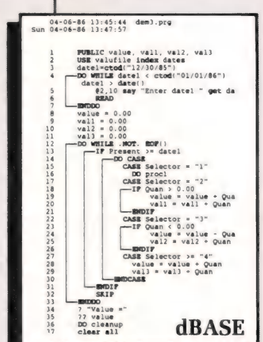
The Index (Cross-Reference list) saves you time by showing exactly where variables are used and where functions, procedures, and routines are called. Locations where new values may be assigned to variables are shown, making it easy to track down that mysterious value change.







$97.00



Structure Outlining solves the problem of hard-to-see nested control structures by automatically drawing lines around them.

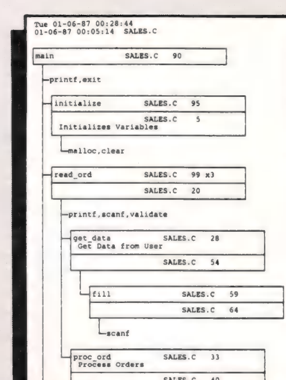
Automatic Indentation of source code and listings reduces your editing time and ensures indentation accuracy.

Plus . . . Source Print generates a table of contents listing functions and procedures. Keywords can be printed in boldface on most printers. Multi-statement BASIC lines can be split for readability. Functions and procedures can be drawn by name from one or more source files to form a new file.

Tree Diagrammer™

shows your program's overall organization at a glance. Ordinary program listings merely display functions, procedures, and subroutines sequentially, but do not display the relationships between these routines. Our revolutionary new Tree Diagrammer automatically creates an "organization chart" of your program showing the hierarchy of calls to functions, procedures, and subroutines. Recursive calls are indicated and designated comments in the source code will appear on the chart.

Tree Diagrammer helps you organize your program more logically. And you'll be amazed at how easy it is to debug when you see how your routines interact.



$77.00

Aldebaran Laboratories 3339 Vincent Rd. Pleasant Hill, CA 94523 415-930-8966

YES! Rush me ☐ Source Print @ $97. ______ ☐ Tree Diagrammer @ $77. ______
☐ Both $155. Ship/Handling $5. For CA add 6% tax ______ Total ______
Name ______
Company ______
Address ______
City ______ State ______ Zip ______
☐ Check enclosed ☐ VISA ☐ MasterCard ☐ American Express
Card # ______ Exp. Date ______
Signature ______ Phone # ______ 58

Integers Don't Float

by Ray Mariella

Calculating square roots without penalties

To do graphics programming, you need a fast, integer square root routine—the faster the better. Unfortunately, many published square root routines seem to be transported floating-point routines, and the demands for accuracy are quite different for floating-point numbers and integers. In fact, a single pass of Newton's method may give you sufficient accuracy and still be faster than the routines borrowed from floating-point experience.

The use of Newton's method is familiar: given a target integer, N, the first step is to get an estimate X_0 of the square root of N, then apply Newton's first derivative method to iterate, which is:

$$f(X_1) = f(X_0) + (X_1 - X_0)(df(X)/dX)$$

If $\sqrt{N}$ is desired, let $f(X) = X \times X$ and $df(X)/dX = 2X$. Then, if X_1 is the square root of N, $f(X_1) = N$. Using this definition of f(X), the equation can be written $N = (X_0 \times X_0) + (X_1 - X_0)(2X_0)$, or to find X_1 from a first guess X_0, use:

$$X_1 = (X_0 + N/X_0)/2$$

If X_1 isn't accurate enough, substitute X_1 for X_0 in this second equation to get X_2, and so on. The catch comes from the implementation of "and so on."

In algorithms that have been transferred from floating-point procedures, in which 64 bits and more are used for accuracy, many iterations may be needed and an error value must be calculated and monitored. When the error is small enough, the routine stops and returns the latest value, X_i, as the square root. Because multiple iterations are needed and an error value is monitored on each pass, the usual code does not use the second equation on its first pass but rather uses $X_1 = N/X_0$ for convenience. You can see this in the typical code fragment for 32-bit integers on the 8086, shown in Example 1, below.

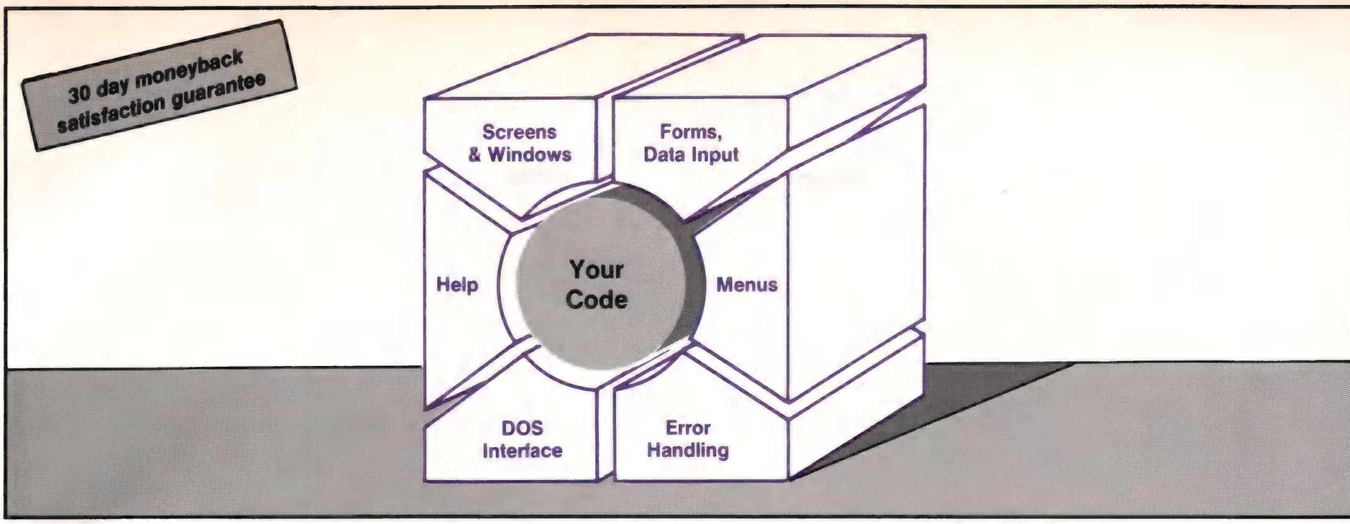
Even with a good first guess of X_0, such as the average of the maximum lower bound and the minimum upper bound by factors of 2, a single pass of ISQRT does not usually suffice. For example, to find the square roots of all the integers from 1 to 65,535, an average of 1.6 divisions per root is needed, or an extra 42,150 iterations. Using the same first guess, ROOT.C (the C code in Example 2, page 50) allows the accurate calculation of the integer square roots using a single pass. The assembly-language code for one pass of Newton's method is much shorter than that for ISQRT, as shown in Example 3, page 52.

At this point it's useful to define accuracy. With most integers, an integer square root is not an exact root. When I speak of the "exact" square root, I refer to rounding the exact floating-point square root to the nearest integer. My assembly-language code is written for the 8086 line of CPUs, and my "exact" answers come from the 8087 numeric coprocessor. (See Listings One through Four, beginning on page

```
; the 32 bit integer N is in DI:SI
; initial guess X0 is in BX
;
ISQRT:      MOV  DX,DI              ;prepare for division
            MOV  AX,SI              ;DX:AX / BX
            DIV  BX                 ;N/X0
            SUB  AX,BX              ;error term
            CMP  AX,1               ;check if > +1
            JG   ISQ1               ;if above 1, keep on
            CMP  AX,\sc0\1          ;check for \sc0\1,0,+1
            JGE  done               ;if OK, get out
ISQ1:       SAR  AX,1               ;(N/X0 \sc0\X0)/2
            ADD  BX,AX              ;(N/X0 +X0)/2 = X1
            JMP short ISQRT         ;use X1 as X0
```

Example 1: *A code fragment for calculating square roots of 32-bit integers on t6*

Ray Mariella, 872 El Quanito Ct., Danville, CA 94526. Ray works at the Lawrence Livermore National Laboratory in Livermore, California, growing compound semiconductor films for ultra-high-speed circuits and optical devices.



C-Worthy Interface Library helps you smoothly pull together all aspects of an excellent Human Interface.

C Programmers: Wrap an Exciting, Bullet-Proof Interface Around Your Code Quickly.

Introducing... C-Worthy® Interface Library

The only human interface package you need. That's what our customers are telling us. One early adopter, Novell, Inc. uses it exclusively in the development of their NetWare® Utilities, which reach over 500,000 users. You see, C-Worthy Interface Library is the only library available to handle every aspect of your program's human interface, all in one package. Now your programs will have a consistent look and feel. You no longer have to integrate pieces of libraries from different manufacturers.

As important as you know users are, you often don't have the time to heavily invest in writing routine code. And that's OK, because with over 400 tight, ready-to-use functions, C-Worthy Interface Library takes care of the tedium and lets you spend your time doing what you enjoy. Concentrate on the heart of your application — features that make it unique, special. Let C-Worthy Interface Library do your:

- Menus
- Error Handling
- DOS Interface
- Context Sensitive Help
- Screens, Windows
- Forms, Data Input (optional)

You control color, size, border, location, etc. And if there's anything you want to change, you can. Source is available to provide you with the flexibility you need. And you can distribute your applications freely, with no royalties.

C-Worthy Interface Library requires hard disk media with 256K RAM. MSDOS 2.0 + and IBM PC, or compatible, TI Professional, NEC APC III, or VICTOR 9000. C-Worthy is a registered trademark of Custom Design Systems, Inc.

Tech Specs

- Compilers: Microsoft 3.0+, Quick, Turbo, Lattice. All models.
- 350+ functions written in C, 75+ in Assembler.
- Menus: Fully support pop-up, Lotus style, MS Windows style (pull-down), pull-up.
- Errors: DOS, program, and user.
- DOS Interface: 62 functions. File handling, dir. and drive management, date & time conversion, wildcards, more.
- Help: System and context sensitive.
- Screens: Screen display, color palettes, save, restore, scroll, more.
- Windows: Exploding, tiled, pop-up, overlapping. Direct video access and virtual. Up to 50 active at any time.
- Keyboard Handling: Regular, function, interrupt, background procedures.
- Editing: String and word wrap text.
- Form Interface Library: 118 functions. Over 15 field types, and user definable field types. 3 levels of data validation: type, multiple field ranges, optional validation procedures. Hide, lock, or secure a field. Optimal field movement.
- Foreign Languages: All text messages in separate files for easy translation.
- Compatible with MS Windows.
- OS/2 special overlay when released.
- Machines: Autodetect for MDA, CGA, EGA, VGA, TI, AT&T, Victor.
- No royalties.

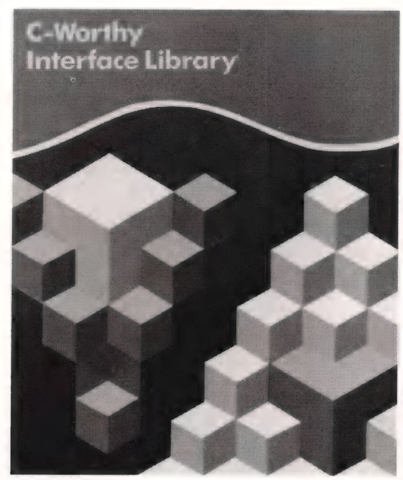
"I heartily recommend this package."
— David A. Schmitt, president, Lattice, Inc.
Over 400 developers in 16 countries already use it.

Thorough Documentation

Indexed alphabetically and by category, the 700+ page Reference Guide includes for each function: an example, description, calling conventions, return values, and related functions. The 250 page User's Guide gets you going with its tutorial and "Getting Started" sections.

CIRCLE 127 ON READER SERVICE CARD

"C-Worthy is a comprehensive C library whose time has come. I heartily recommend it as your next purchase." —Computer Language, 8/87



C-Worthy Interface Library:

Object only	**$ 195**
Form Interface Library add-on	**$ 100**
Object with Forms	**$ 295**
Object with Forms & Library Source	**$ 495**

Please specify compiler and version when ordering.

To Order Call
(800) 821- 2492
in MA (617) 337-6963

Solution Systems™
541-D Main Street, Suite 410
South Weymouth, MA 02190

INTEGERS DON'T FLOAT

(continued from page 48)

98.)

Applications vary, and if you demand that the integer square root either agree with the exact root or differ by –1, then ROOT.C works up to N = 127,087 (the square root of 127,088 is 356.49 and a single pass gives 357). If you're willing to accept 0, or +1, or –1 as an error, then a single pass of Newton's method can be used up to 1,941,799 (for 1,941,800 the square root is 1393.48 and a single pass gives 1,395).

```
/*  ROOT.C a square root algorithm by RPM  */
/*   long integers, single pass of Newton  */

   #include <stdio.h>

   main (\sc128\)
   {

   long int N, guess2, sqrrt;
   register int infi, guess1;

       printf ( "\n square root of what number " );
       scanf ("%ld",&N);
     {  guess1 = infi =1;
          guess2 = N;
   logit:  infi <<= 1;
          if ( infi < guess2 )
          {  guess2 >>= 1;           /* div by 2 */
             guess1 = infi;
             goto logit;
          }
          guess1 += guess2;          /* sum */
          guess1 >>= 1;              /* avg */
       /* newton's method  */
          infi = N / guess1;
          sqrrt = infi + guess1;
          sqrrt >>= 1;
       }
          printf ( " square root = %ld", sqrrt);
    }
```

Example 2: *C code to calculate integer square roots using a single pass*

Skipping all those compare, conditional jump statements, and multiple passes can save a lot of CPU time. On my 8-MHz PC6300 with V30, the assembly-language program RALL16 (Listing Four) which is optimized for 16-bit integers, finds all the square roots from 1 to 65,535 in 2.5 seconds, with an empty loop time of 0.2 seconds, or about 35 microseconds per root. This is slightly faster than the speed at which the 8087 performs the same task. I found that 29,776 of the roots were 1 less than the exact root and the rest agreed with the exact root. The Microsoft C 4.0 version of RALL16 took 4.6 seconds for all integers from 1 to 60,000, or about 73 microseconds per root.

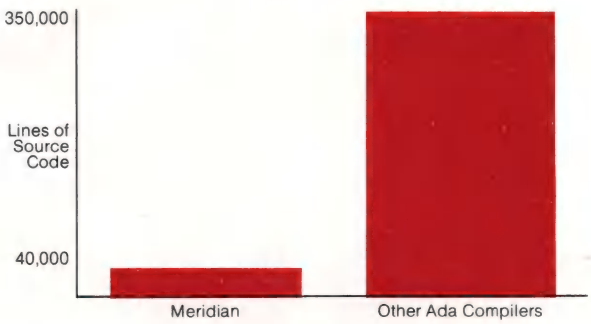
For 32-bit integers up to 4,294,967,295 (FFFF:FFFF), two passes of the second equation are needed. My program code, ISQRT32.ASM (in

THE ADA® WORLD HAS CHANGED.™

Next Generation Ada Technology

Third generation Ada is here today with Meridian AdaVantage, and it's validated by the Department of Defense and their suite of 2,700 tests. Ada's first generation was the proving ground technology of the early eighties. Then came the large validated compilers of the mid-eighties, which were often inefficient and machine specific. Now Meridian offers a compact efficient Ada technology that is highly portable from PC's to minis to mainframes.

COMPACT MERIDIAN Ada IMPLEMENTATION



The chart above depicts the dramatic difference between Meridian's implementation and other comercially available Ada implementations. Meridian's order of magnitude smaller code size results in an extremely fast compiler that will produce highly optimized code.

Meridian's Track Record

Since 1981, we've been building compilers the old-fashioned way, through hard work and experience gained from our development of successful portable compiler products. These products are installed at over 1,500 sites, including almost all major DoD contractors as well as commercial software developers.

Price/Performance Breakthrough

The Meridian AdaVantage v2.0 validated Ada compiler costs $795 and provides a production quality Ada development tool.

SIEVE BENCHMARK

	Meridian v2.0	Alsys v1.3
Compile and link time	27 sec	59 sec
Execution time	4.6 sec	4.9 sec
Execution size	27,344 bytes	42,129 bytes
Price	$795	$2,995

NOTE: All times measured on an IBM at 5170 (8MHz) and a 4MB RAM card required by the Alsys system. When running without the RAM card, the Meridian compile and link time is 46 seconds.

The Meridian AdaStarter incorporates all of the features of AdaVantage, with certain size limitations. AdaStarter is perfect for anyone who wants to learn how to program in Ada. The complete $99 cost is applicable toward the purchase price of the AdaVantage production compiler.

The compilers all run in a standard PC configuration with 640K of memory, a hard disk, and DOS v2.1 or higher.

To order today, or get more information, call 1-800-221-2522 (outside California) and 1-714-380-9800 (inside California).

"The Meridian compiler is a very well-thought-out compiler. The compiler is fast and execution speed is more than adequate... Overall, we'd rate the Meridian compiler as very solid."

— COMPUTER LANGUAGE, DECEMBER 1986

"The more affordable AdaVantage v1.0 is good for the average programmer because of its price, the extent of its implementation, and its relaxed hardware requirements... [AdaVantage v2.0] should be competitive with Alsys Ada, since it will be a full Ada at less than a third of the price of Alsys."

BYTE, JULY 1987

VALIDATED Ada

THIS PRODUCT CONFORMS TO ANSI/MIL-STD-1815A AS DETERMINED BY THE AJPO UNDER ITS CURRENT TESTING PROCEDURES

MERIDIAN SOFTWARE SYSTEMS, INC.

23141 Verdugo Drive, Suite 105, Laguna Hills, CA 92653
800/221-2522 (outside CA) 714/380-9800 (inside CA)
Telex: 650-268-0547 MCI Fax: 714/380-1683

CIRCLE 130 ON READER SERVICE CARD

INTEGERS DON'T FLOAT

(continued from page 50)

Listing One), for the 8086 line is cumbersome, but it does work for all these integers (it takes approximately 153 hours to compare the V30 and 8087 square roots). Again, the result after two passes of Newton's method either agrees with the exact integer or is 1 less. This code executes in about 110 microseconds per root on my machine. (Here the 8087 is more than twice as fast as the V30 and looks very attractive indeed!)

Again with Microsoft C 4.0, the two-pass version of ROOT.C takes an average of 360 microseconds per root for integers from 10,000 to 100,000,000 in steps of 10,000 (using register variables). This V30 performance is about the same speed as the same program on a Macintosh Plus with Lightspeed C. The maximum integer that ROOT.C can handle depends upon the compiler and its acceptance of such things as unsigned register variables.

The decision whether to use the single-pass or double-pass version of Newton's procedure can usually be made by the programmer before starting, based on the number of pixels on the screen and thus the maximum integer that the program will see. In either event, my measurements have shown that using the unaltered Newton's method provides greater speed and somewhat better accuracy than can be attained using the more common algorithms that came from the floating-point literature.

```
; the 32 bit integer N is in DI:SI
; initial guess X0 is in BX
;
NEWTON:     MOV   DX,DI         ;prepare for division
            MOV   AX,SI         ;DX:AX / BX
            DIV   BX            ;N/X0
            ADD   BX,AX         ;(N/X0 +X0)/2 = X1
            RCR   AX,1          ;(N/X0 \sc0\X0)/2
```

Example 3: *Assembly-language code for one pass of Newton's method*

Along the same lines, in the March 1986 issue of DDJ, *Richard Campbell described both a C routine and an assembly-language version for the NS320xx processor for calculating square roots.—eds.*

Availability

Most of the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

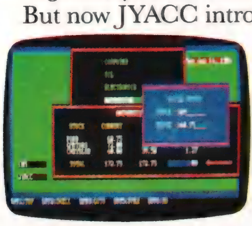
(Listings begin on page 98.)

Vote for your favorite feature/article.
Circle Reader Service **No. 4.**

Our front end helps protect your back end.

Today's users require sophisticated interfaces for their applications. Yet complex front ends are a real pain to create, especially when the specs change and your deadlines don't.

But now JYACC introduces JAM™, a powerful user interface development tool that makes it easier than ever to design and revise your complex applications.



Use windows and colors freely with JAM.

JAM is the first tool that does it all, from prototyping to implementation. With JAM, you start by creating screens and linking them together to develop an application shell. You can experiment with the look of the interface, and even explore "what if" scenarios on the application flow. Then you attach processing routines, and your application is complete.

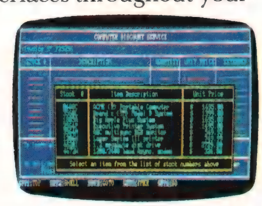
JAM works under the following operating systems:

- UNIX®
- MS-DOS®
- VMS®
- XENIX®
- RMX™
- VOS™

You'll be amazed at how quickly your applications spring to life with JAM's interactive screen management utility. You can use features like context-sensitive help, shifting and scrolling fields, a variety of visual attributes and extensive data validations to create exciting screens, windows and menus—all without writing a single line of code. JAM also lets you test your prototypes at any time.

JAM lets you draw from its extensive subroutine library to help you write processing routines faster. And revisions to your applications are easier, because your subroutines are insulated from the data entry and presentation details of the interface.

JAM is extremely portable. It runs on almost every computer, from PCs to superminis, and works under 6 operating systems. This allows you to develop consistent interfaces throughout your company—a significant asset to the Fortune 500 companies that have been using JAM for more than a year.



Enhance applications with context-sensitive help.

JAM from JYACC. It gets your front ends—and back ends—in great shape. Call for more information about JAM and our demo diskette. **800-458-3313**

JYACC FORMAKER™, JAM's screen manager, is also available separately.

JYACC, Inc., 116 John Street
New York, NY 10038 212-267-7722

Introducing JAM

JYACC Application Manager. *The Composer for Sophisticated Applications.*

JYACC

Excellence in Systems & Application Design

MS-DOS, XENIX: Microsoft Corp.; UNIX: AT&T Bell Labs; RMX: Intel Corp.; VMS: Digital Equipment Corp.; VOS: Stratus Corp.

CIRCLE 132 ON READER SERVICE CARD

A Graphics Toolbox for Turbo C—Part 2

by Kent Porter

Easy menu bars, pop-up windows, and pull-down submenus

In Part 1 of "A Graphics Toolbox for Turbo C" (*DDJ*, November 1987), I developed a library of low-level graphics routines for Turbo C and explored ways to incorporate them into high-level drawing routines for APA (all-points addressable, or pixel-oriented) graphics. In this article, I use the library routines developed in Part 1 to create text graphics such as menu bars, pop-up windows, pull-down submenus, and the like.

Popping up a visual object that obscures a portion of the screen, then restoring that overlaid area after removing that object, is a visually exciting if relatively simple programming trick. Before writing an object to screen, you simply copy display memory into a save buffer; to make the object go away—that is, to restore the screen to its former appearance—you just write back the screen previously saved to buffer into display memory. In this way, the text obscured by the pop-up reappears magically intact.

Finding display memory can be a bit more challenging, however. The display memory location depends on the kind of adaptor you're using. To get that information from the operating system, I constructed a function called *videomode( )*. It determines which type of video adaptor is active: if *videomode( )* returns 7, there's an MDA with display memory at *B000h:0*; otherwise, display memory begins at segment *B800h*.

Text screens for IBM-standard adaptors occupy 4,096 bytes, whereas for the Hercules the display memory is 16,384 bytes. You can find out the video buffer size from *40h:4Ch* using Turbo C's *peek( )* function, which returns an integer indicating screen memory size.

Some non-MDA adaptors provide for more than one video page. That is, they subdivide the total display memory into 4K segments (pages) and allow you to select which is the active display buffer. That's what the video library function *setpage( )* does, and *activepage( )* tells which it is. Therefore, with a non-MDA adaptor, you have to determine the active page and multiply by 4,096 to determine the starting offset of the current video buffer within segment *B800h*. (Or you can fetch this number directly from *40h:4Eh*.)

Once you know the size and location of the active page's video buffer, you can allocate a node of that size on the heap and use Turbo C's *movedata( )* function to save the display image. You can then create a pop-up. Later, simply reverse the source and destination parameters in *movedata( )* to restore the screen and make the pop-up vanish.

Restoring the screen is problematic with the IBM-brand CGA, which has a flawed design that generates "snow." This occurs when you write directly to screen memory in text mode. Fortunately, this isn't a problem with the Compaq and most non-IBM CGA clones. The solution with the IBM board involves synchronizing character movements with the video controller. (For more detail on this procedure, see Ray Duncan's *Advanced MS-DOS*, page 79).

The *saveScrn( )* and *restScrn( )* functions near the end of POPWIN.I (Listing One, page 106) furnish somewhat simplified versions of this discussion. They assume that the active page is 0—the default—and that an IBM-standard adaptor is in use. Also, they don't deal with snow.

You might wish to take a more sophisticated approach to saving screens. In the demo developed for this article, I never save more than one screen image, and thus I simply copy the active screen buffer to and from an object of the same size on the heap.

Real-world situations often have a hierarchy of pop-ups, in which one leads to another, to another, and so on. In that case, you build a stack (LIFO) structure on the heap so that you can retreat back down the hierarchy in an orderly manner. A circular doubly linked list is ideal for this purpose.

In summary, then, the steps for handling pop-ups are:

- Save the display buffer on the heap.
- Write the pop-up.
- On a signal from the user, copy the heap image back to the display buffer, thus restoring the screen.

Kent Porter, 1909-4 Montecito Rd., Mountain View, CA 94043. Kent has written 17 books about programming and hundreds of magazine articles on computer hardware and software. He is a technical editor for DDJ.

Now that we've solved one of the big mysteries surrounding pop-ups, let's discuss the abstraction of complexity.

Visual Objects as Data Structures

Although the screen is a free-form area that can contain literally anything you want it to, special objects such as pop-ups and menu bars always conform to a prescribed set of attributes. That is, they have a defined number, size, location, color combination (foreground/background), border, text content, and so on.

The point is that any object can be defined in terms of two characteristics: its appearance and its text content. Sometimes it's appropriate to combine them into one, and sometimes it's not. I'll consider both cases.

This approach suggests the use of data structures and accompanying routines that interpret those structures on the display. On that basis, you can describe visual objects by building structures and leave it to the routines to translate them into reality. Let's prove the point with a discussion of menu bars.

Building Menu Bars

The menu bar has become a staple of interactive software. You see it everywhere: Reflex, Paradox, Lotus 1-2-3, Turbo C itself. It's that row of selections across the top of the screen, telling you the categories of things you can do with the program. Usually, when you pick a selection, that choice leads to others via another menu bar (the Lotus approach) or a pull-down submenu, which I discuss later in this article.

In software with a consistent user interface, the menu bar always has a predefined place on the screen. Here I assume it's at the top—row 0—but your own application might dictate another location, in which case you'll have to modify the *menubar()* routine to suit. A menu bar also has a fixed text content. If you want to add or delete selections, that's a different menu.

The advantage of a routine such as *menubar()* (Listing Two, page 106) is that it interprets the structure passed to it. This enables you to pass different structures to a common function, obtaining the desired results in each case. The previous menu bar is replaced by the new one, effecting an instantaneous change in context.

Given that all menu bars in an application occupy the same display row, any given menu bar can be described in terms of its color scheme, number of selections, and text content. That leads to the definition of a menu bar specification

With multiple pop-up windows, a circular doubly linked list is ideal for screen saving.

as the C structure:

```
typedef struct {
   int background, foreground; /* colors */
int nsels; /* no. of selections */
   char *sel; /* pointer to text content */
} MENUBARSPEC;
```

You might want to use different color schemes to identify various menu bar levels: for example, top level = red background, middle level = magenta, and low level = blue. Also, it's likely that each menu will have its own number of potential selections. A generalized routine such as *menubar()* can easily handle these variations, simply plucking them from the structure.

The *menubar()* function first identifies its environment by getting the active page and the number of text columns available (40 or 80), and then it builds the background/foreground attribute byte with a call to the video library's *chattr()* function (a service function that does not call the ROM BIOS directly or indirectly). Next, it clears the current menu bar row and sets the attribute by writing out a stream of spaces for the full screen width. Finally, it writes the text of the menu selections.

The menu selections are specified in a separate string, a pointer to which appears in the menu specification. For convenience, the string has the form:

```
menutext = "sel1\0sel2\0sel3\0seln\0"
```

in which each text element is null-terminated. This is easier to handle than the more conventional two-dimensional array of characters, as the last loop in *menubar()* illustrates.

Using this approach, you can define a menu bar by first initializing its text content, then its appearance and a pointer to the text content in a structure of type *MENUBARSPEC*. If you had two menu levels, separately defined, you could display the top level with *menubar (&firstLevel);* and, on the appropriate signal from the user, switch to the next with *menubar (&secondLevel);*.

Because menu bars tend to have a fixed appearance and content, it's easy to implement them using this scheme. Pop-ups are more flexible and consequently a little more demanding. Still, you can create a wide variety of pull-down menus and pop-up windows using a common set of routines and structures, as discussed next.

Pop-Ups and Pull-Downs

In practice, there's no difference between the two. A pop-up is a window that can appear anywhere on the display. A pull-down is a pop-up that looks as though it dropped down from a menu bar selection. The distinction is purely one of location.

Pop-ups rely on the ROM BIOS window routine (function 6 under interrupt *10h*). Despite its comprehensive-sounding name, the BIOS routine is useful for only two things: scrolling a subset of the screen and filling that subset with a background attribute to give it a color. User-written routines that want to do more than this will have to provide code to overcome its limitations.

POPWIN.I (Listing One) furnishes generalized routines for managing

GRAPHICS TOOLBOX
(continued from page 55)

pop-up windows, relying on the low-level routines from the video library. This is an *#include* file, and there's little advantage to making it into a linkable library because any windowing program needs all the functions it contains.

Like a menu bar, a pop-up can be described in terms of its characteristics, which in this case are:

- the coordinates of its opposite corners (giving both location and size)
- the text attribute (background/foreground colors)
- the border type (single- or double-score)

The *POPDESCR* structure in POPWIN.I provides a window descriptor.

The static array *bord[ ][ ]* gives the ASCII values for single- and double-score text characters that form the window border. The border choices are 0 for no border and 1 and 2 for single- and double-scores, respectively.

POPWIN.I contains four functions for directly manipulating windows:

- *popMake( )* creates and displays the blank window described by the structure passed to it as an argument
- *popScroll( )* scrolls the window's contents upward by one row
- *popxy( )* positions the cursor relative to the window's upper-left corner
- *popPuts( )* writes a string starting at the specified cursor position

The other two functions—*saveScrn( )* and *restScrn( )*—save and restore the display image.

To use these routines, you must first initialize a *POPDESCR* structure for each window you plan to use. Thereafter, simply pass a structure address to the pop routines as you call them. Because the routines do not preserve the caller's cursor position, your code must do so before the calls.

Note that, unlike the menu bar structure, *POPDESCR* doesn't contain a pointer to the window's text contents. This allows you to use a common descriptor for several windows that contain variable text: context-sensitive help windows, for example. Use *popPuts( )* to write to the window.

The *popMake( )* function opens a window, fills it with the assigned text attribute, draws a border around it (unless the structure's border member is zero), and positions the cursor in the upper-left corner. When creating a new window, call this function first. Note that the border is outside the window's defined text area; that is, if the window's upper-left corner is at (12, 15), the left side of the box is at column 11 and the top is at row 14. The right side and bottom are similarly one unit beyond the text area. Thus, don't put a window at any extreme position on the display, lest the border be partly invisible or partially visible in an unexpected place.

The *popScroll( )* routine shifts the window's contents upward one row by calling the *winScroll( )* function in the video library, passing it the boundaries of the area to be scrolled and the text attribute to fill the newly opened row at the bottom. *PopPuts( )* uses this routine to guard against text overrunning the bottom of the window.

Use *popxy( )* to position the cursor within the window, using the upper-left corner as the origin. No matter where the window is physically located, *popxy(0,0,win)* refers to its upper-left corner, effecting viewport text coordinates. This function is therefore a window-oriented analog to the video library's *gotoxy( )*.

The *popPuts( )* function writes text to the window, starting at the specified coordinates with respect to the window's upper-left corner. It contains safeguards to ensure that the text does not get outside the window, wrapping if the text tries to go beyond the right side and scrolling if it tries to go past the bottom. The function accommodates a new line ('\n') embedded in the text, but it does not have formatting capabilities like *printf( )* does. If you want to write variables to a window, use *sprintf( )* to set up the text string, then pass the results to *popPuts( )*.

The functions for saving and restoring the video buffer work unpre-

PROGRAMMER BUYS INBOARD 386; ENDS UP IN HOT WATER.

When Jim Johnson's C program got to be more than his IBM PC AT® could handle, he had two choices:

Come up with $6000 for a brand new 386 system.

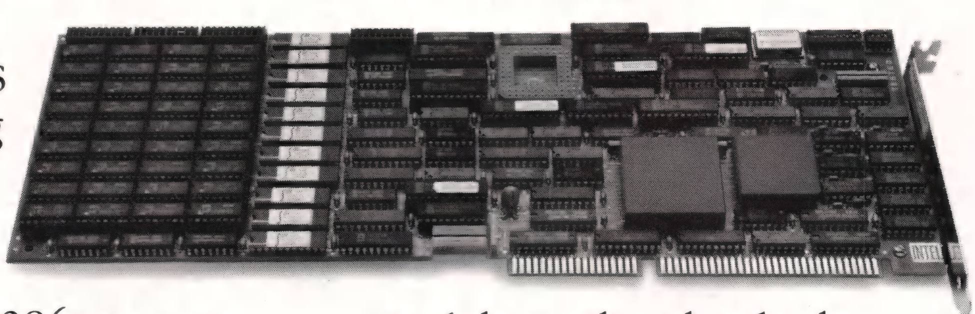
Or spend ~~$2000~~ $1595 for an Intel Inboard™ 386 system.

As you might suspect, he chose the Intel Inboard 386. And got the full power of a 386 system.

Allowing him to cut his compiling time in half. Without compromising one bit on reliability. And with the extra ~~$4000~~ $4400, he put a mahogany hot tub on his deck.

How can we give you the same for so much less? Simple. If you have an IBM PC AT or compatible, you already own 2/3 of a 386 system. And when you install an Inboard 386, you get the rest of it.

Besides the price, how do they compare? According to a recent *InfoWorld* product review, "Inboard has the best computing speed of all micro systems we have tested to date," including all other 386 systems.



The reason Inboard is so fast is because of its zero wait state cache and 32 bit memory. To give you even greater performance, there's also a special socket for the 80387-16 math coprocessor.

Software compatibility is unsurpassed as well. According to *PC Week*, "The Inboard 386 proved perfectly compatible with a standard IBM PC AT and every software product we tested." It's compatible with advanced software, too, including 386 control software for multitasking. And a number of developers are already using Inboard to create OS/2® applications.

Of course, you~~'re~~ probably wondering if a ~~$2000~~ $1595 system can be as reliable as a $6000 one. Absolutely. Because it's built by the same company that designed the 80386 microprocessor. And backed by a five-year warranty.

If you still need more information on which system is best, call us at (800) 538-3373. Or contact your local dealer.

And then take a hot bath. When your workload is too much to handle, it's the best way we know to unwind.

intel®

CIRCLE 135 ON READER SERVICE CARD

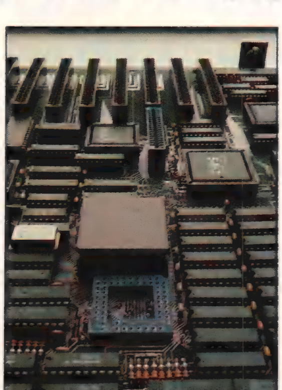
SAVE YOUR PC FROM EARLY RETIREMENT.

GET HAUPPAUGE'S NEW 386 MOTHERBOARD.

386 SPEED–ONLY $1,495

Give your PC a new lease on life! With our industry first 386 MotherBoard, your PC, PC/XT or compatible will revel in speeds equal to the Compaq DeskPRO 386. And faster. Because we've built in 1 Megabyte of high speed RAM and a 387 math coprocessor socket for speeds that will knock you off your rocker.

To keep retirement at bay, our 386 MotherBoard is compatible with the PC/AT (BIOS and I/O) — allowing you to run the new generation of DOS, OS/2. We've also included a 16-bit expansion slot that accommodates the latest I/O expansion card. No accelerator card can give you so much versatility.



With 386 power and true AT software compatibility, your business, desktop publishing and engineering applications will get a boost to boast about! **Technical Features** ▪ 16 MHz 80386 ▪ 1 Megabyte of 100 nsec 4-way interleaved RAM ▪ PC/AT compatible I/O and BIOS for support of OS/2 ▪ Seven 8-bit expansion slots ▪ Two 16-bit expansion slots ▪ One 32-bit RAM expansion slot ▪ Optional 16 MHz 80387 math coprocessor ($695)

Put the power of the 386 into your IBM PC for 1/4 the cost of a 386 computer. And put off your PC's retirement. For more information on our easy-to-install Motherboard, call **1 (800) 443-6284.** In New York, call **(516) 360-3827.**

Hauppauge Computer Works, Inc.
358 Veterans Memorial Highway,
Commack, New York 11725

Hauppauge!

CIRCLE 136 ON READER SERVICE CARD

(continued from page 56)

dictably with IBM display adaptors (MDA, CGA, and EGA) but work normally with Hercules and other third-party video boards. You can make them more intelligent with the techniques in last month's installment. As written, these functions assume that if the video mode is 7, an MDA (or an EGA emulating an MDA) is present and using screen memory at segment *B000* and that otherwise a color device is using the video buffer at segment *B800*.

In either case, because these routines service text displays, they assume a 4,096-byte display memory. *SaveArea( )* allocates a node of that size on the heap and copies the video buffer to it, returning a pointer to the node. Afterward, you can create a pop-up. To make the pop-up disappear later, pass the node pointer to *restScrn( )*, which moves the saved image back into the video buffer and frees the heap space.

Now let's put these two visual subsystems—menu bars and pop-ups—to work in a demonstration.

Tying It Together

MENUDEMO.C (Listing Three, page 106) writes a menu bar across the top of the screen and provides two pull-down submenus on successive keypresses. Some text appears under the pull-downs so that you can see how overlaid information is restored when a window is removed from the screen.

The pull-downs are described by the *filePop* and *editPop* structures and their text contents by the strings *fileMenu* and *editMenu*. For convenience, the strings contain embedded new lines that will be processed by *popPuts( )* to give the pull-downs their proper appearance.

The initialization steps in *main( )* use the video library's *chattr( )* function to complete the *filePop* structure, then copy the structure to *editPop*. The location of *editPop* is computed with reference to the second entry on the menu bar, and the size of the box is changed. The program is now ready to run.

Setmode(3) has no effect on an MDA, but it places other adaptors in color mode (an EGA with a monochrome monitor displays the colors as intensities). After clearing the screen, the program writes the menu bar and prints out some information about the display.

The program uses *oldx* and *oldy* to retain the cursor position before reporting it. This is so that, later, a subsequent output line will overlay the cursor position report.

When the user presses a key, the first pull-down appears, created by *popFileMenu( )*. On the second keypress, the file pull-down disappears, the background is restored, and *popEditMenu* flashes up the edit pull-down. Two additional keypresses restore the original screen and end the program.

Note that, except for the structure initialization, it takes only one statement to display a menu bar and only two to create a pop-up (*popMake( )* and *popPuts( )*). These calls exercise a good deal of underlying code, of course, but this code remains out of sight in the *#include* files, where it doesn't needlessly clutter the program's listing.

Full Circle

Obviously, the graphics library I've built here wasn't intended to be a full-blown graphics toolkit; rather, it was meant to provide you with a skeletel framework to build upon. Use the techniques illustrated here to build more powerful visual subsystems in Turbo C that satisfy your specific application development needs.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listings begin on page 106.)

Vote for your favorite feature/article.
Circle Reader Service **No. 5.**

Listing One *(Listing continued, text begins on page 30.)*

```
Listing 1

/* cache header */

typedef struct CACHE {
  int      recl;             /* record length */
  int      maxr;             /* number of records in cache */
  long     hits;             /* number of true finds for cacfind() */
  long     miss;             /* number of misses for cacfind() */
  long     adds;             /* number of calls to cacnew() */
  int      (*proc)();        /* pointer to function for processing */
  long     idnt;             /* identifier passed to proc() */
  long     *nums;            /* numbers of records */
  char     *recs;            /* pointer to first record */
  short    *next;            /* array of next pointers */
  short *prio;               /* array of prior pointers */
  char     *mark;            /* array of process-record markers */
  short    lru;              /* oldest index */
  short mru;                 /* newest index */
  } CACDS;

long      cacamem;  /* amount of memory allocated */

CACDS     *cacallo(int, int, int ( *)(), long);
char      *cacmem (int);
char      *cacnum (struct CACHE  *, long);
char      *cacold (struct CACHE  *);
int        cacflsh(struct CACHE  *);
int        cacnew (struct CACHE  *, short);
char      *cacfind(struct CACHE  *, long);
int        cacproc(struct CACHE  *, long);
int        cacunpc(struct CACHE  *, long);
int        cacstat(struct CACHE  *, long  *, long  *, long  *);
int        cacfree(struct CACHE  *);
```

End Listing One

Listing Two

```
Listing 2

/* cache.c - memory cache

  alan deikman 12/86

  These routines handle a memory cache of fixed sized records.
  All memory is allocated through the standard library routine
  malloc().

  Each routine other than cacallo() takes as the first parameter
  a character pointer that is originally returned by the cacallo()
  routine.

  cacallo()          Allocate cache
  cacold()           Get oldest record
  cacnum()           Number oldest record and make it the newest
  cacflsh()          Process all marked records
  cacfind()          Find record n and make it newest
  cacproc()          Mark record for processing when freed
  cacunpc()          Unmark record for processing when freed
  cacstat()          Get cache statistics
  cacfree()          Free cache

  */

#include <stdio.h>
#include <malloc.h>
#include "cache.h"

/* cacallo() - Allocate cache */

CACDS *cacallo(num, recl, extf, idnt)
int       num;               /* number of records to allocate */
int       recl;              /* length of each record */
int       (*extf)();         /* pointer to external processing function */
long      idnt;              /* parameter passed to external function */
{
  CACDS    *cac;
  char     *cacmem();
  int      i = 0;

  /* set up cac structure with initial values */

  if (num < 2) num = 2;
  cac = (CACDS *) cacmem(sizeof(CACDS));
  cac->recl = recl;
  cac->maxr = num;
  cac->proc = extf;
  cac->idnt = idnt;
  cac->hits =
  cac->miss =
  cac->adds = 0L;
  cac->nums = (long *)  cacmem(num * sizeof(long));
  cac->next = (short *) cacmem(num * sizeof(short));
  cac->prio = (short *) cacmem(num * sizeof(short));
  cac->mark = cacmem(num);
  cac->recs = cacmem(num * recl);

  /* initial lru/mru chain */

  while (i < num) {
    cac->next[i] = i + 1;
    cac->prio[i] = i - 1;
    cac->nums[i] = -1;
    cac->mark[i] = 0;
    i++; }
```

Listing Two *(Listing continued, text begins on page 30.)*

```
        cac->next[num - 1] = cac->prio[0] = -1;
        cac->mru = 0;
        cac->lru = num - 1;

        /* return cache pointer */

        return cac; }


    /* allocate memory with error checking */

    char *cacmem(siz)
    int       siz;
    {
      char *c;
      c = malloc(siz);
      if (c == (char *) 0) {
        fprintf(stderr, "cacallo:  Can't allocate memory.\n");
        fprintf(stderr, "Tried to get %d bytes on top of %ld bytes already\n",
                siz, cacamem);
        exit(1); }
      cacamem += siz;
      return c; }

    /* number oldest record and make it the newest.  if the record was
       marked for exit procesing, call external processing function.
       return pointer to record */

    char *cacnum(cac, num)
    CACDS     *cac;                 /* cache header */
    long      num;                  /* number new MRU record */
    {
      char    *rec = cac->recs + (cac->lru * cac->recl);

      /* call external function */

      if (cac->mark[cac->lru] && cac->proc)

        (*(cac->proc))(cac->idnt, cac->nums[cac->lru], rec);

      /* unmark record and make it newest */

      cac->mark[cac->lru] = 0;
      cac->adds++;
      cac->nums[cac->lru] = num;
      cacnew(cac, cac->lru);

      /* return record; ready for usage */

      return rec; }
```

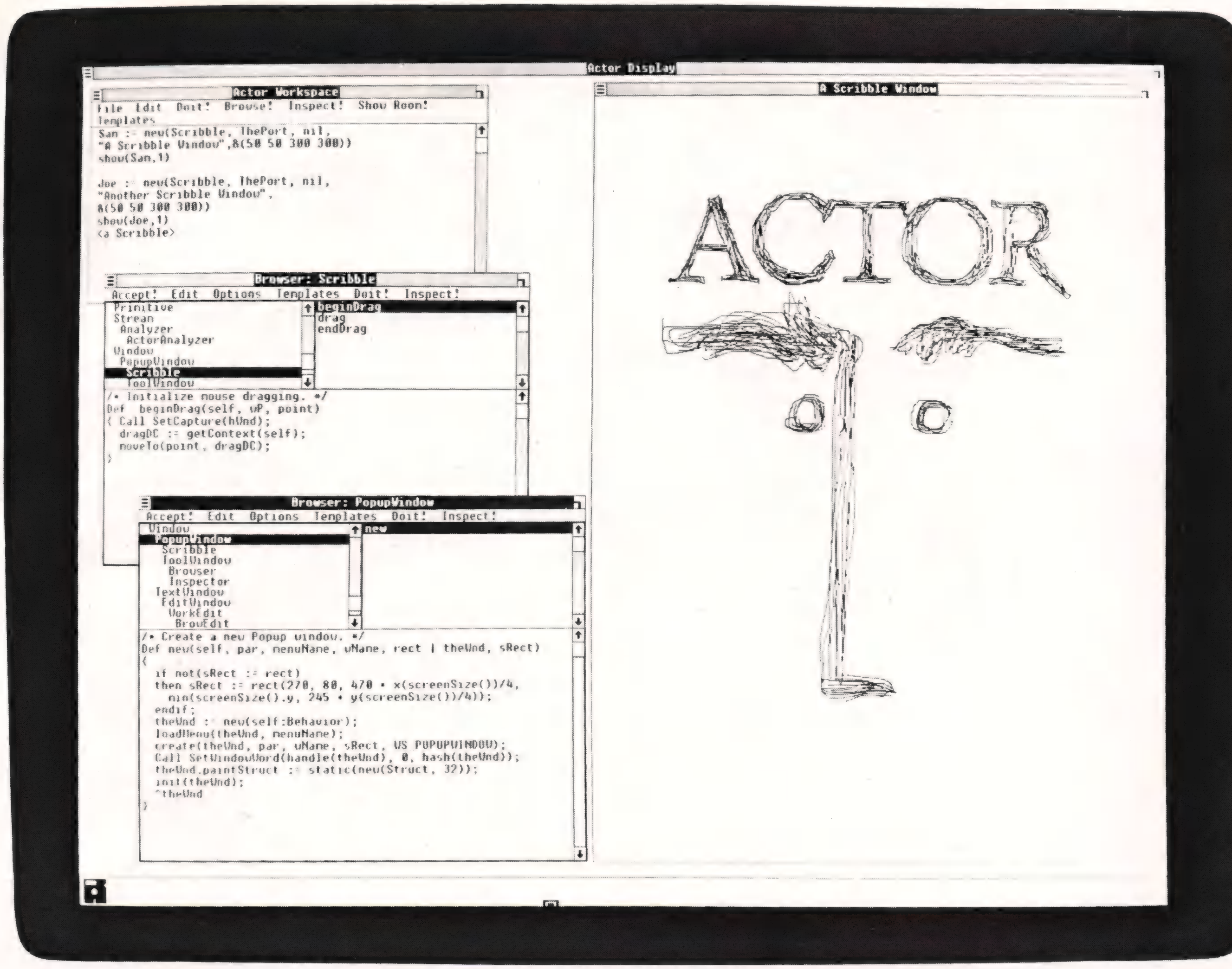
(continued on next page)

Listing Two *(Listing continued, text begins on page 30.)*

```
/* get pointer to oldest record without altering age */

char *cacold(cac)
CACDS       *cac;               /* cache header */
{
  return cac->recs + (cac->lru * cac->recl); }


/* if an exit processing routine has been defined, process all marked
   records */

cacflsh(cac)
CACDS       *cac;               /* cache header */
{
  int       i;
  char      *rec;

  if (!(cac->proc)) return;

  for (i = 0; i < cac->maxr; i++) if (cac->mark[i]) {
    rec = cac->recs + (i * cac->recl);
    (*(cac->proc))(cac->idnt, cac->nums[i], rec);
    cac->mark[i] = 0; }

  return; }


/* make record newest */

cacnew(cac, rec)
CACDS       *cac;               /* cache header */
short       rec;                /* record to make newest */
{

  /* if this record is already the newest, just return */

  if (rec == cac->mru) return;

  /* change prior's next */

  cac->next[cac->prio[rec]] = cac->next[rec];

  /* change next's prior - if there was no next that means this was the
     lru record.  change lru */

  if (cac->next[rec] != -1)
    cac->prio[cac->next[rec]] = cac->prio[rec];
  else {
    if (rec != cac->lru) {
      fprintf(stderr, "cacnew: panic\n");
      exit(1); }
    cac->lru = cac->prio[rec]; }

  /* now the record is out of the chain.  stick it back in at the mru end. */

  cac->prio[cac->mru] = rec;
  cac->next[rec]      = cac->mru;
  cac->prio[rec]      = -1;
  cac->mru            = rec;

  /* done */

  return; }

/* find record and make it newest */

char *cacfind(cac, num)
CACDS       *cac;               /* cache header */
long        num;                /* record number to look for */
{
  int i;

  for (i = 0; i < cac->maxr; i++) if (cac->nums[i] == num) {
    cac->hits++;
    cacnew(cac, i);
    return cac->recs + (i * cac->recl); }

  cac->miss++;
  return (char *) 0; }

/* mark record for external processing */

cacproc(cac, num)
CACDS       *cac;               /* cache header */
long        num;                /* record to mark */
{
  int i;

  for (i = 0; i < cac->maxr; i++) if (cac->nums[i] == num) {
    cac->mark[i] = 1;
    return; }

  return; }

  /* un-mark record for external processing */

  cacunpc(cac, num)
  CACDS       *cac;               /* cache header */
  long        num;                /* record to unmark */
  {
    int i;

    for (i = 0; i < cac->maxr; i++) if (cac->nums[i] == num) {
      cac->mark[i] = 0;
      return; }

    return; }

  /* get statistics of the cache */
```

(continued on page 67)

HOW TO WRITE A WINDOWS APPLICATION IN TEN MINUTES.



Actor™ is a new language that combines Microsoft® Windows with object-oriented programming. This means you can produce mouse and window applications very quickly.

For example, we created a simple "paint" program, and used it to draw the Actor logo you see on the screen. The whole program only took ten lines and ten minutes. Part of it is in the middle window on the left.

Above, you see the commands that initialized the paint window and made it appear on the screen. Below, some code that's built into Actor, specifying window behavior. Through a process known as "inheritance," it's called into play automatically.

Try programming in this new way, and you'll never go back.

Find out about Actor: Call The Whitewater Group,® (312) 491-2370.

Technology Innovation Center
906 University Place, Evanston, IL 60201

MICROSOFT WINDOWS

CIRCLE 140 ON READER SERVICE CARD

Programmer's Paradise Gives You Superb Selection, Personal Service and Unbeatable Prices!

Welcome to Paradise. The microcomputer software source that caters to your programming needs.

Discover the Many Advantages of Paradise...

- Lowest price guaranteed
- Latest versions
- Huge inventory, immediate shipment
- Knowledgeable sales staff
- Special orders
- 30-day money-back guarantee

Over 500 brand-name products in stock—if you don't see it, call!

We'll Match Any Nationally Advertised Price.

	LIST	OURS
386 SOFTWARE		
ADVANTAGE 386 C	895	**799**
ADVANTAGE 386 PASCAL	895	**799**
MICROPORT SYSTEM V/386 (COMPLETE) *SPECIAL*	799	**679**
MICROSOFT WINDOWS/386	195	**125**
PHARLAP 386\ASM/LINK	495	**419**
PHARLAP 386 DEBUG	195	**155**
SCO XENIX SYS V 386 (COMPLETE)	1495	**1195**
VM/386 *SPECIAL*	195	**119**
ARTIFICIAL INTELLIGENCE		
ARITY STANDARD PROLOG	95	**CALL**
GOLDEN COMMON LISP	495	**CALL**
MICROSOFT LISP	250	**155**
PC SCHEME	95	**85**
SMALLTALK V	99	**85**
TURBO PROLOG	100	**65**
TURBO PROLOG TOOLBOX	100	**65**
ASSEMBLERS/LINKERS		
ADVANTAGE DISASM. *SPECIAL*	295	**249**
ADVANTAGE LINK	395	**359**
ASMLIB	149	**125**
EZ_ASM	70	**65**
MS MASM *CALL–REBATE OFFER*	150	**95**
PASM86	195	**109**
PLINK86PLUS	495	**275**
RELMS CROSS ASSEMBLERS	CALL	**CALL**
UNIWARE CROSS ASSEMBLERS	CALL	**CALL**
VISIBLE COMPUTER 80286	100	**89**
BASIC		
DB/LIB	99	**89**
FLASH-UP	89	**79**
MACH 2	75	**59**
MS QUICKBASIC *CALL–REBATE OFFER*	99	**65**
QUICKPAK	69	**59**
TRUE BASIC	100	**69**
TURBO BASIC	100	**65**
DATABASE TOOLBOX	100	**65**
EDITOR TOOLBOX	100	**65**
TELECOM TOOLBOX	100	**65**

XENIX/UNIX PRODUCTS

	LIST	OURS
MICROPORT & SCO	CALL	**CALL**
ADVANTAGE C++	695	**CALL**
BTRIEVE	595	**455**
C-TERP	498	**379**
INFORMIX ESQL/C	749	**CALL**
INFORMIX 4GL	1500	**CALL**
INFORMIX SQL	995	**CALL**
KORN SHELL	125	**115**
MICROSOFT LANGUAGES	CALL	**CALL**
PANEL	625	**535**
PANEL PLUS	795	**675**
REAL TOOLS	149	**89**
RM/COBOL	1250	**949**
RM/FORTRAN	750	**549**
SCO MULTIVIEW (286)	395	**319**
SCO MULTIVIEW (386)	495	**399**

	LIST	OURS
C++		
ADVANTAGE C++	495	**479**
PFORCE++	395	**209**
C COMPILERS		
C86PLUS	497	**375**
HIGH C	595	**CALL**
LATTICE C *SPECIAL PRICE*	500	**CALL**
MICROSOFT C	450	**269**
QUICKC *CALL–REBATE OFFER*	99	**65**
TURBO C	100	**65**
C INTERPRETERS		
C-TERP	298	**219**
INSTANT C	495	**369**
RUN/C	120	**79**
RUN/C PROFESSIONAL	250	**155**
C LIBRARIES		
BASIC_C	175	**129**
C ASYNCH MANAGER	175	**135**
C-FOOD SMORGASBORD	150	**95**
W/SOURCE CODE	300	**179**
C/PAC (PFORCE, PRE-C) *SPECIAL*	495	**279**
C TOOLS PLUS/5.0 *NEW*	129	**99**
C UTILITY LIBRARY	185	**119**
C-XPERT	295	**249**
ESSENTIAL COMMUNICATIONS	185	**119**
COMMUNICATIONS PLUS	250	**189**
GREENLEAF C SAMPLER *SPECIAL*	95	**69**
GREENLEAF COMM LIBRARY	185	**125**
GREENLEAF FUNCTIONS	185	**125**
MULTI-C	149	**135**
PFORCE	395	**209**
RESIDENT C W/SOURCE	198	**149**
TIMESLICER *SPECIAL*	295	**249**
W/SOURCE CODE	1000	**895**
TURBO C TOOLS	129	**99**
COBOL		
FPLIB	149	**129**
MICRO FOCUS COBOL PRODUCTS	CALL	**CALL**
MICROSOFT COBOL	700	**439**
MICROSOFT SORT	195	**129**
OPT TECH SORT	149	**99**
REALCICS	995	**785**
REALIA COBOL	995	**785**
W/REALMENU	1145	**899**
RM/COBOL	950	**759**
RM/COBOL-85	1250	**999**
RM/SCREENS	395	**315**
SCREENIO	400	**379**
DEBUGGERS		
ADVANCED TRACE-86	175	**119**
C-SPRITE	175	**119**
PERISCOPE I	345	**289**
PERISCOPE II	175	**139**
PERISCOPE III 8 MHZ	995	**825**
PERISCOPE III 10 MHZ	1095	**899**
PFIX 86 PLUS	395	**209**
T-DEBUG PLUS	60	**49**
XVIEW86	60	**49**
DISK/DOS/KEYBOARD UTILITIES		
BACK-IT	100	**89**
BOOKMARK	100	**79**
COMMAND PLUS	80	**69**
DISK OPTIMIZER	60	**55**
INTELLIGENT BACKUP	150	**135**
NORTON COMMANDER	75	**55**
ADVANCED NORTON UTILITIES	150	**89**
PDISK	145	**89**
FILE MANAGEMENT		
BTRIEVE	245	**185**
XTRIEVE	245	**185**
REPORT OPTION	145	**99**
BTRIEVE/N	595	**455**
XTRIEVE/N	595	**455**
REPORT OPTION/N	345	**269**
C-TREE	395	**315**
R-TREE	295	**239**
C-TREE/R-TREE BUNDLE	650	**519**
DBC III	250	**169**
DBC III PLUS	750	**595**
INFORMIX PRODUCTS	CALL	**CALL**
PHACT MANAGER	249	**219**
EDITORS		
BRIEF	195	**CALL**
W/DBRIEF	275	**CALL**
CVUE W/SOURCE CODE	250	**199**
EDIX	195	**155**
EMACS	295	**265**
EPSILON	195	**149**
KEDIT	125	**99**
LMK	195	**139**
PC/EDT	250	**229**
PC/VI	149	**109**
Pi EDITOR *SPECIAL, NEW*	195	**149**
PMATE	195	**109**
SPF/PC	195	**145**
VEDIT PLUS	185	**129**
XTC	99	**79**
FORTRAN COMPILERS		
LAHEY FORTRAN	477	**CALL**
LAHEY PERSONAL FORTRAN 77	95	**89**
MICROSOFT FORTRAN	450	**269**
RM/FORTRAN	595	**399**
FORTRAN UTILITIES/LIBRARIES		
DOCUMENT'ER	129	**115**
DIFF-E-Q	495	**445**
FORTLIB	125	**109**
FORTRAN ADDENDA	165	**139**
GRAFLIB	175	**169**
GRAFMATICS/PLOTMATICS	135	**119**
MATHPAC	495	**445**
NO LIMIT	129	**115**
SPINDRIFT LIBRARY *NEW*	149	**135**
SSP/PC	350	**269**
GRAPHICS		
ADVANTAGE GRAPHICS (C)	250	**225**
ESSENTIAL GRAPHICS	250	**185**
GSS GRAPHIC DEV. TOOLKIT	495	**375**
HALO	300	**205**
HALO (5 MICROSOFT LANG.)	595	**389**
METAWINDOW PLUS	275	**229**
TURBOWINDOW/C	95	**79**
TURBO HALO (FOR TURBO C)	100	**79**
LINT		
PC-LINT	139	**99**
PRE-C	295	**155**
MODULA-2		
FARBWARE MODULA-2	90	**79**
LOGITECH MODULA-2		
COMPILER PACK	99	**79**
DEVELOPMENT SYSTEM	249	**CALL**
TOOLKIT	169	**139**
WINDOW PACKAGE	49	**39**
ROM PACKAGE AND CROSS RUNTIME DEBUGGER	299	**239**
REPERTOIRE	89	**75**

	LIST	OURS
OPERATING SYSTEMS		
MICROPORT SYSTEM V/AT (COMPLETE)	549	**465**
SCO XENIX SYSTEM V (COMPLETE)	1295	**995**
WENDIN-DOS	99	**79**
OTHER MICROPORT, SCO, WENDIN PRODUCTS	CALL	**CALL**
PASCAL COMPILERS		
MARSHAL PASCAL *SPECIAL*	189	**149**
MICROSOFT PASCAL	300	**185**
PASCAL-2 *SPECIAL*	350	**299**
TURBO PASCAL *NEW V. 4.0*	100	**65**
TURBO PASCAL DEV. LIB. *NEW*	395	**259**
BORLAND ADD-ONS	CALL	**CALL**

FEATURED PRODUCTS

PRO-C 'C' — Source Program Generator. It produces stand-alone 'C' program exactly to your specification. Each program is fully optimized and ready to run. They are complete with documented source code and full system documentation. PRO-C gives freedom from the time-consuming work associated with Source Code development, allowing concentration on more demanding tasks such as system design and specification.
List: $399 **Special Price: $379**

CARBON COPY PLUS — Communications software, that features both Remote Control and Terminal Emulation.
Remote Control: Use your PC to observe and control a remote PC as though the two were one. Access your office PC while you are at home.
Terminal Emulation: Includes V-52, V-100, TVI-920, IBM-3101. Easy to access host computers and on-line information databases. Carbon Copy Plus will also allow you to send and receive telexes.
List: $195 **Special Price: $139**

Pascal-2 — Highly optimized Pascal compiler, with source level debugger, profiler.
List: $350 **Special Price: $299**

ADVANTAGE Disassembler — Provides immediate feedback as you work, strong results in tables on disk. Final output is ready for MS assembler. Handles COM and EXE files 8086/186/286 code and 8087/287 coprocessors.
List: $295 **Special Price: $249**

	LIST	OURS
TURBO PASCAL ADD-ONS		
ALICE	95	**69**
DOS/BIOS & MOUSE TOOLS	75	**69**
FLASH-UP	89	**79**
METRABYTE DATA ACQ. TOOLS	100	**89**
SCREEN SCULPTOR	125	**95**
SYSTEM BUILDER	150	**129**
IMPEX	100	**89**
REPORT BUILDER	130	**115**
TURBO ASM	99	**69**
T-DEBUG PLUS	60	**49**
TURBO ASYNCH PLUS *NEW*	129	**99**
TURBO EXTENDER	85	**65**
TURBO HALO	129	**99**
TURBO MAGIC	99	**89**
TURBO MASTER	125	**99**
TURBO OPTIMIZER	75	**65**
TURBO POWER TOOLS PLUS *NEW*	129	**99**
TURBO POWER UTILITIES	95	**79**
TURBO WINDOW/PASCAL	95	**79**
SCREEN DISPLAY/WINDOWS		
C-SCAPE	279	**CALL**
CURSES W/SOURCE CODE	250	**169**
GREENLEAF DATA WINDOWS	225	**155**
W/SOURCE CODE	395	**289**
HI-SCREEN XL	149	**119**
JYACC FORMAKER	995	**449**
JYACC JAM	750	**669**
MICROSOFT WINDOWS	99	**65**
MS WINDOWS DEVELOPMENT KIT	500	**309**
PANEL PLUS	495	**395**
PANEL FOR QUICKC/TURBO C	129	**95**
QUICKSCREEN	195	**175**
SCREEN ACE *SPECIAL*	195	**159**
SCREENSTAR W/SOURCE	198	**155**
VITAMIN C	225	**159**
VC SCREEN	99	**79**
WINDOWS FOR DATA	295	**235**
VIEW MANAGER	275	**199**
ZVIEW	245	**169**

HARDWARE PRODUCTS

	LIST	OURS
AMDEK 722 MONITOR	750	**499**
AMDEK 730 MONITOR	899	**569**
AST ADVANTAGE PREMIUM W/512K	495	**319**
AST RAMPAGE! 286 W/512K	545	**349**
HERCULES GRAPHICS CARD PLUS	299	**195**
HERCULES IN COLOR CARD	499	**329**
IRMA 2	1195	**779**
ORCHID TURBO EGA	749	**495**
ORCHID TURBO PGA	1495	**1099**
VEGA DELUXE	379	**259**

	LIST	OURS
ADDITIONAL PRODUCTS		
ADVANTAGE VCMS	379	**329**
BASTOC	495	**CALL**
CARBON COPY PLUS *SPECIAL*	195	**139**
DAN BRICKLIN'S DEMO PROGRAM	75	**59**
DB2C	299	**CALL**
FLOW CHARTING II	229	**205**
MAGIC PC	195	**179**
MKS TOOLKIT	139	**115**
NORTON GUIDES	100	**65**
PFINISH	395	**225**
POLYMAKE	149	**125**
POLYTRON PVCS	CALL	**CALL**
SOURCE PRINT	95	**75**
TP2C	249	**199**
TREE DIAGRAMER	77	**69**

Terms and Policies
- **We honor MC, VISA, AMERICAN EXPRESS** No surcharge on credit card or C.O.D. Prepayment by check. New York State residents add applicable sales tax. Shipping and handling $3.00 per item, sent UPS ground. Rush service available, prevailing rates.
- Programmer's Paradise will match any current nationally advertised price for the products listed in this ad.
- Prices and Policies subject to change without notice.
- Hours 9AM EST — 7PM EST

*Ask for details. Some manufacturers will not allow returns once disk seals are broken.

Corporate Buyers — Call for special discounts and benefits!

1-800-445-7899
In NY: 914-332-4548
Customer Service: 914-332-0869
International Orders: 914-332-4548
Telex: 510-601-7602

Happy Holidays from

Programmer's Paradise™

A Division of Hudson Technologies, Inc.
42 River Street, Tarrytown, NY 10591

Listing Two *(Listing continued, text begins on page 30.)*

```
cacstat(cac, hit, mis, add)
CACDS      *cac;                    /* cache header */
long       *hit, *mis, *add; /* values to return */
{
  *hit = cac->hits;
  *mis = cac->miss;
  *add = cac->adds;
  return; }

/* free cache */

cacfree(cac)
CACDS      *cac;                    /* cache header */
{
  cacflsh(cac);
  cacamem -= sizeof(CACDS) + (cac->maxr * sizeof(long)) +
             (cac->maxr * 2 * sizeof(short)) + (cac->maxr) +
             (cac->maxr * cac->recl);
  free(cac->recs);
  free(cac->prio);
  free(cac->next);
  free(cac->nums);
  free(cac);

  return; }
```

End Listing Two

Listing Three

```
Listing 3

#include <stdio.h>

#include "cache.h"

CACDS *cache;

main() {

  printf("Cache test routine\n");
  cache = cacallo(8, 128, (char *) 0, 1L);
  printf("Memory allocated = %ld\n\n",  cacamem);

  while (ctest()) cprint();

  exit(0); }


int ctest() {

  int opt, rec;
  long num;

  printf("\n1-old, 2-num, 3-find, 4-proc: ");
  scanf("%d", &opt);

  switch (opt) {
    case 0: return 0;
    case 1: printf("cacold returns %lx\n", cacold(cache));  return 1;
    case 2: printf("enter record: ");
               scanf("%ld", &num);
               printf("cacnum returned %lx\n", cacnum(cache, num));
               return 1;
    case 3: printf("input record to find: ");
            scanf("%ld", &num);
            printf("cacfind returned %lx\n", cacfind(cache, num));
            return 1;
    case 4: printf("input record to process: ");
            scanf("%ld", &num);
            cacproc(cache, num);
            printf("cacproc called\n");
            return 1;
    otherwise: return 1; }
  return; }

cprint() {

  register int i;

  printf("Cache print:  hits=%ld  miss=%ld  adds=%ld\n",
         cache->hits, cache->miss, cache->adds);

  printf("Block Numbers Next  Prior Mark  LRU=%d  MRU=%d\n", cache->lru, cache->mru);
  printf("----- ------- ----- ----- -----\n");
  for (i = 0; i < cache->maxr; i++)
    printf("%5d %7ld %5d %5d %4d\n", i, cache->nums[i],
                                                  cache->next[i],
                                                  cache->prio[i],
                                                  cache->mark[i]);
  return; }
```

End Listings

Listing One *(Text begins on page 38.)*

BOOT.C

```
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>

#include "loc.h"
#include "externs.h"


void  create_bootstrap(seg_list, entry)
SEG_DESCRIPTOR *seg_list ;
unsigned char  *entry ;
{
    unsigned int   count ;
    unsigned char  *ptr ;

    SEG_DESCRIPTOR *p, *q ;

    /*
       This function sets up a new class which contains the bootstrap
       code to the program entry point.  The bootstrap segment is
       always located at physical address FFFF0H is ROMable.

       The bootstrap record is appended to the load module for the
       purpose of locate processing.
    */

    /* Traverse the linked list to the end */
    p = seg_list ;
    while (p->next != NULL)
       p = p->next ;

    /* Allocate the memory for the bootstrap record */
    if ((q = (SEG_DESCRIPTOR *) malloc(sizeof (*p))) == NULL)  {
       perror(__FILE__) ;
       exit(1) ;
    }

    /* Append the bootstrap record to the end of the list */
    p->next = q ;

    /* Initialize the bootstrap segment descriptor */
    strcpy(q->name, "??BOOT") ;
    strcpy(q->class, "(ABSOLUTE)") ;
    q->vseg = 0xffff ;
    q->pseg = 0xffff ;
    q->offset = 0x0000 ;
    q->len = 5 ;
    q->inited = TRUE ;
    q->romable = TRUE;
    q->symbols = 0 ;
    q->symbol_list = NULL ;
    q->next = NULL ;

    /* Allocate RAM and build the reset vector code using a far jump */
    ptr = malloc(q->len) ;
    *ptr = 0xEA ;
    *((unsigned char **)(ptr + 1)) = entry ;

    /* Append the bootstrap code on tail of the load module */
    q->position = lseek(tmp_file, 0L, SEEK_END) ;
    count = write(tmp_file, ptr, q->len) ;
    if (count != q->len)    {
       perror(__FILE__) ;
       exit(1) ;
    }

    free(ptr) ;
    return ;
}
```

End Listing One

Listing Two

FILES.C

```
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <string.h>

#include "loc.h"
#include "externs.h"

#define  F_OPEN       O_RDONLY | O_BINARY
#define  F_CREATE     O_CREAT | O_TRUNC | O_RDWR | O_BINARY


void  open_file_system(input_file)
char  *input_file ;
{
    char  *create_str = "Can't create %s" ;
    char  *open_str = "Can't open %s" ;
    char  errmsg[MAX_LINE] ;
    char  *filename_ext ;

    /*
       This module is responsible for openning or creating all of the
       files used by this utility.
    */

```

(continued on page 73)

DOS LOCATE UTILITY

Listing Two

(Listing continued, text begins on page 38.)

```
    /* Perform all the filename processing */
    strcpy(module_name, strupr(input_file)) ;
    strcpy(exe_fname, module_name) ;
    strcat(strcpy(map_fname, exe_fname), ".MAP") ;

    if ((config == FALSE) || (strlen(config_fname) == 0))
        strcat(strcpy(config_fname, exe_fname), ".CFG") ;
    else
        strupr(config_fname) ;

    if ((hex_name == FALSE) || (strlen(abs_fname) == 0))
        strcat(strcpy(abs_fname, exe_fname), ".HEX") ;
    else
        strupr(abs_fname) ;

    strcat(strcpy(print_fname, exe_fname), ".LOC") ;
    strcat(exe_fname, ".EXE") ;

    /* Create the temporary file used for segment fixups and
                                                location */
    strcpy(tmp_fname, "LOCATE.$$$") ;
    tmp_file = open(tmp_fname, F_CREATE, S_IWRITE) ;
    if (tmp_file == -1)    {
        sprintf(errmsg, create_str, tmp_fname) ;
        perror(errmsg) ;
        exit(1) ;
    }

    /* Create the absolute output file */
    abs_file = open(abs_fname, F_CREATE, S_IWRITE) ;
    if (abs_file == -1)    {
        sprintf(errmsg, create_str, abs_fname) ;
        perror(errmsg) ;
        exit(1) ;
    }

    /* Open the .EXE file */
    exe_file = open(exe_fname, F_OPEN) ;
    if (exe_file == -1)    {
        sprintf(errmsg, open_str, exe_fname) ;
        perror(errmsg) ;
        exit(1) ;
    }

    /* Create the locate map output file */
    print_file = fopen(print_fname, "wt") ;
    if (print_file == NULL)    {
        sprintf(errmsg, open_str, print_fname) ;
        perror(errmsg) ;
        exit(1) ;
    }

    /* Open the configuration file for reading */
    config_file = fopen(config_fname, "rt") ;
    if (config_file == NULL)    {
        sprintf(errmsg, open_str, config_fname) ;
        perror(errmsg) ;
        exit(1) ;
    }

    /* Open the linker map file for reading */
    map_file = fopen(map_fname, "rt") ;
    if (map_file == NULL)    {
        sprintf(errmsg, open_str, map_fname) ;
        perror(errmsg) ;
        exit(1) ;
    }

    return ;
}


void    close_file_system()
{
    char   errmsg[MAX_LINE] ;
    char   *close_str = "Unable to close %s" ;
    char   *delete_str = "Unable to delete %s" ;

    /*
       This function is responsible for shutting down the file system
       and cleaning up the temporary files.  All files opened for
       reading are closed and all files open for writing are closed
       (normal exit) and possibly deleted (control-C abort event).
    */

    /* Close the link map */
    if (fclose(map_file) != 0)    {
        sprintf(errmsg, close_str, map_fname) ;
        perror(errmsg) ;
    }

    /* Close the locate configuration file */
    if (fclose(config_file) != 0)    {
        sprintf(errmsg, close_str, config_fname) ;
        perror(errmsg) ;
    }

    /* Close the .EXE file */
    if (close(exe_file) == -1)    {
        sprintf(errmsg, close_str, exe_fname) ;
        perror(errmsg) ;
    }
```

(continued on next page)

Listing Two *(Listing continued, text begins on page 38.)*

```

    /* Close the locate map */
    if (fclose(print_file) != 0)    {
       sprintf(errmsg, close_str, print_fname) ;
       perror(errmsg) ;
    }

    /* Close the absolute or hex object module */
    if (close(abs_file) == -1)    {
       sprintf(errmsg, close_str, abs_fname) ;
       perror(errmsg) ;
    }

    if (user_abort == TRUE)    {
       /* Delete the locate map */
       if (remove(print_fname) == -1)    {
          sprintf(errmsg, delete_str, print_fname) ;
          perror(errmsg) ;
       }

       /* Delete the object file */
       if (remove(abs_fname) == -1)    {
          sprintf(errmsg, delete_str, abs_fname) ;
          perror(errmsg) ;
       }
    }

    /* Close and then delete the temporary file */
    if (close(tmp_file) == -1)    {
       sprintf(errmsg, close_str, tmp_fname) ;
       perror(errmsg) ;
    }

    if (remove(tmp_fname) == -1)    {
       sprintf(errmsg, delete_str, tmp_fname) ;
       perror(errmsg) ;
    }

    return ;
}
```

End Listing Two

Listing Three

LOADEXE.C

```
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <dos.h>

#include "loc.h"
#include "externs.h"


char  *warn_str = "Warning: Unable to locate virtual segment %04X\n" ;

char  *load_exe_file()
{
    char  buf[128] ;
    int   count, i ;
    long  seek_pos ;
    unsigned int   module_size, pseg, *reloc_ptr, segment ;
    unsigned int   read_size, mem_size ;
    char  *load_addr, *entry, *str ;

    EXE_HEADER  header;
    SEG_DESCRIPTOR *p ;

    /*
       This function reads in the .EXE file and performs the fixup of any
       segment references.
    */

    /* Read in the .EXE file header information */
    count = read(exe_file, (char *) &header, sizeof(header)) ;
    if (count != sizeof(header))    {
       perror(__FILE__) ;
       exit(1) ;
    }

    /* Exit if not a valid .EXE file */
    if (header.signature != 0x5A4D)    {
       perror("Not an .EXE file") ;
       exit(1) ;
    }

    /* Seek to the start of the load module */
    if (lseek(exe_file, (long) header.header_size * 16, SEEK_SET) == -1L) {
       perror(__FILE__) ;
       exit(1) ;
    }

    /* Compute how much memory can be allocated for reads */
    mem_size = 32 * 1024 ;
    if (mem_size > _memavl())
       mem_size = _memavl() ;

    /* Allocate the memory */
    if ((load_addr = malloc(mem_size)) == NULL)    {
```


(continued on page 76)

Listing Three *(Listing continued, text begins on page 38.)*

```
        perror(__FILE__) ;
        exit(1) ;
    }

    while (1)    {
        /* Read in a segment of the load module */
        read_size = read(exe_file, load_addr, mem_size) ;
        if (read_size == 0)    {
            free(load_addr) ;
            break ;
        }

        /* Write it back out to the temporary file */
        count = write(tmp_file, load_addr, read_size) ;
        if (count != read_size)    {
            perror(__FILE__) ;
            exit(1) ;
        }
    }

    /* Find the relocation list */
    if (lseek(exe_file, (long) header.first_reloc_item, SEEK_SET) == -1L) {
        perror(__FILE__) ;
        exit(1) ;
    }

    /* Perform the segment fixups on the temporary file */
    for (i = 0; i < header.reloc_items; i++)    {
        /* Read in a relocation item */
        count = read(exe_file, (char *) &reloc_ptr, sizeof(reloc_ptr)) ;
        if (count != sizeof(reloc_ptr))    {
            perror(__FILE__) ;
            exit(1) ;
        }

        /* Compute the position of the fixup in the temporary file */
        seek_pos = (long) FP_SEG(reloc_ptr) ;
        seek_pos = seek_pos * 16 + FP_OFF(reloc_ptr) ;
        if (lseek(tmp_file, seek_pos, SEEK_SET) == -1L)    {
            perror(__FILE__) ;
            exit(1) ;
        }

        /* Read in the virtual segment from the fixup */
        count = read(tmp_file, (char *) &segment, sizeof(segment)) ;
        if (count != sizeof(segment))    {
            perror(__FILE__) ;
            exit(1) ;
        }

        /* Perform the location */
        if (locate_virtual_segment(segment, &pseg) == ERROR)
            fprintf(stderr, warn_str, segment) ;

        segment = pseg ;

        /* Re-seek back to the fixup */
        if (lseek(tmp_file, seek_pos, SEEK_SET) == -1L)    {
            perror(__FILE__)   ;
            exit(1) ;
        }

        /* Write the physical segment number to the fixup */
        count = write(tmp_file, (char *) &segment, sizeof(segment)) ;
        if (count != sizeof(segment))    {
            perror(__FILE__) ;
            exit(1) ;
        }
    }

    /* Process the program entry point */
    if (locate_virtual_segment(header.code_seg_disp, &pseg) == ERROR)
        fprintf(stderr, "Warning: Unable to locate entry point\n") ;

    FP_SEG(entry) = pseg ;
    FP_OFF(entry) = header.initial_pc ;

    return   entry ;
}
```

End Listing Three

Listing Four

LOCATE.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>

#include "loc.h"
#include "globals.h"
#include "externs.h"

/*
    LOCATE *** MS-DOS ROM Utility
    Copyright (C) 1987 Rick Naro.  All rights reserved.
*/

int   main(argc, argv)
int argc;
```


(continued on page 78)

Listing Four *(Listing continued, text begins on page 38.)*

```
char *argv[];
{
   char  *s, *input_file ;
   unsigned char  *entry_point ;
   int   i ;

   /*
      This is the root module of the locate utility and it controls the
      sequencing of the entire location process.
   */

   /* Install a Control-C interrupt handler */
   if (signal(SIGINT, break_handler) == (int(*)()) -1)   {
      fprintf(stderr, "Failure to install break handler\n") ;
      abort() ;
   }

   /* Build a command line string using argv[0] through argv[argc-1] */
   command_line[0] = '\0' ;
   for (i = 0; i < argc; i++)    {
      strcat(command_line, argv[i]) ;
      strcat(command_line, " ") ;
   }

   /* Test if the user needs help in running this utility */
   if (argc == 1)
      help = TRUE ;

   config_fname[0] = abs_fname[0] = print_fname[0] = '\0' ;

   /* Process each argument in sequence until all are processed */
   while (--argc > 0 && (*++argv)[0] == '-')    {
      for (s = argv[0] + 1;  *s != '\0';  s++)    {
         switch (*s)    {
            case 'b':
               boot_rec = TRUE ;
               break ;

            case 'c':
               config = TRUE ;
               if (*++s)
                  strcpy(config_fname, s) ;
               *s-- = '\0' ;
               break ;

            case 'h':
               hex_name = TRUE ;
               if (*++s)
                  strcpy(abs_fname, s) ;
               *s-- = '\0' ;
               break ;

            default:
               help = TRUE ;
               argc = 0 ;
               break ;
         }
      }
   }
   input_file = argv[0] ;

   if (help == TRUE)    {
      fprintf(stderr, "\nUsage is\n\n") ;
      fprintf(stderr, "\tlocate switches exefile\n\n") ;
      fprintf(stderr, "The valid switches are:\n\n") ;
      fprintf(stderr, "\t%-14s create bootstrap record\n", "-b") ;
      fprintf(stderr, "\t%-14s configuration filename\n", "-c[name]") ;
      fprintf(stderr, "\t%-14s hex filename\n", "-h[name]") ;
      exit(1) ;
   }

   fprintf(stderr, "MS-DOS Locate Utility - Version 1.0\n") ;
   fprintf(stderr, "Copyright (C) 1987 Rick Naro.  ") ;
   fprintf(stderr, "All rights reserved\n\n") ;

   /* Open and create the files used in the location process */
   open_file_system(input_file) ;

   /* Install the routine to shutdown the utility gracefully in the
      event of an error. */
   onexit(close_file_system) ;

   /* Build the segment descriptor list using the link map */
   seg_list = build_seg_list();

   /* Process the locate configuration file */
   if (process_locate_file(seg_list) == ERROR)    {
      fprintf(stderr, "Error(s) reading the locate map\n") ;
      exit(1) ;
   }

   /* Convert any public symbols to their new physical addresses */
   read_symbol_table(seg_list) ;

   /* Read the load module and perform the segment fixups */
   entry_point = load_exe_file() ;

   /* Add a bootstrap record if enabled on the command line */
   if (boot_rec == TRUE)
      create_bootstrap(seg_list, entry_point) ;

   /* Output the load module in the specified format */
   output_hex_OMF(abs_file, seg_list, entry_point) ;

   /* Make the locate map containing the new segment assignments */
```

```
    print_statistics(map_fname, print_fname, command_line, exe_fname, \
       abs_fname, config_fname, entry_point) ;

    exit(0) ;
}


void  break_handler()
{
    /*
       The break handler is provided to catch Ctrl-C interrupts from the
       user and perform a shutdown of the program in a graceful manner.
    */

    /* Set the user abort flag for the file system close routine */
    user_abort = TRUE ;
    exit(1) ;
}

```

End Listing Four

Listing Five

MISC.C

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <dos.h>
#include <errno.h>

#include "loc.h"
#include "externs.h"

extern   int   errno ;

char  *get_mem(size)
unsigned long  size ;
{
    union REGS  regs ;
    char  *p ;

    /*
       This function is a substitute for allocation of huge arrays.  It
       uses DOS system calls to directly allocate up to 64K for a memory
       block.
    */

```

(continued on next page)

Listing Five *(Listing continued, text begins on page 38.)*

```
    regs.x.bx = (unsigned int) (size / 16 + 1) ;
    regs.h.ah = 0x48 ;
    FP_SEG(p) = intdos(&regs, &regs) ;
    if (regs.x.cflag)    {
       errno = ENOMEM ;
       p = NULL ;
    }
    else
       FP_OFF(p) = 0 ;

    return   p ;
}


void  free_mem(p)
char  *p ;
{
    union REGS  regs ;
    struct   SREGS sregs ;

    /*
       This function is the complement of get_mem() in that it releases
       any memory previously acquired with get_mem().
    */

    sregs.es = FP_SEG(p) ;
    regs.h.ah = 0x49 ;
    intdosx(&regs, &regs, &sregs) ;

    return ;
}


int assign_physical_segment(class, seg)
char      *class ;
unsigned int   seg ;
{
    int    error = ERROR ;
    SEG_DESCRIPTOR *p ;

    /*
       This function assigns the specified class name the physical
       segment number.  The first segment within a named class will
       have an offset of zero.  All other segments have a segment
       offset relative to the first segment in the class.
    */

    p = seg_list ;
    while (p != NULL)    {
       if (strcmp(p->class, class) == 0)    {
          p->pseg += seg ;
          p->inited = TRUE ;
          error = OK ;
       }
       p = p->next ;
    }
    return   error ;
}


int   get_next_segment(pclass, cclass, seg)
char  *pclass ;
char  *cclass ;
unsigned int    *seg ;
{
    int    error = ERROR ;
    BOOLEAN  found = FALSE ;
    SEG_DESCRIPTOR *p, *q, *last ;

    /*
       This function returns the next physical segment address
       available for use by CCLASS (current class) after PCLASS
       (previous class).  A typical use is to force the concatenation
       of independent classes.
    */

    /* Search the class list for the occurrence of the CCLASS */
    p = q = seg_list ;
    while (q != NULL)    {
       if (strcmp(q->class, cclass) == 0)    {
          found = TRUE ;
          break ;
       }
       q = q->next ;
    }

    if (found == FALSE)
       return   ERROR ;            /* Error if it can't be found */

    /* Search for PCLASS and then to the end of PCLASS. */
    while (p != NULL)    {
       if (strcmp(p->class, pclass) == 0)    {
          last = p ;
          while(strcmp(p->class, pclass) == 0)    {
             last = p ;
             p = p->next ;
          }

          /* Return the next available segment and adjust the segment
             value for an overflow if necessary. */

```

```
          *seg = last->pseg + last->len / 16 ;
          if (q->offset < (last->len + last->offset) % 16)
             *seg += 1 ;

          return   OK ;
       }
       p = p->next ;
    }
    return   ERROR ;
}


int   dup_class(old_class, new_class)
char  *old_class ;
char  *new_class ;
{
    int   error = ERROR ;
    SEG_DESCRIPTOR *p, *q, *prev, *head ;

    /*
       Copies the contents of the OLD_CLASS entry to the newly created
       NEW_CLASS entry.
    */

    p = seg_list ;
    while (p != NULL)   {
       if (strcmp(p->class, old_class) == 0)    {
          prev = head = NULL ;
          while (p != NULL)    {
             if (strcmp(p->class, old_class) != 0)
                break ;

             /* Create the new segment descriptor */
             if ((q = (SEG_DESCRIPTOR *) malloc(sizeof (*q))) == NULL) {
                perror(__FILE__) ;
                exit(1) ;
             }

             if (prev == NULL)
                head = q ;
             else
                prev->next = q ;

             /* Copy the contents and add the new entry to the list */
             *q = *p ;
             strcpy(q->class, new_class) ;
             q->next = NULL ;

             prev = q ;
             if (p->next == NULL)
                break ;
             else
                p = p->next ;
          }
          while (p->next != NULL)
             p = p->next ;

          p->next = head ;
          return   OK ;
       }
       p = p->next ;
    }
    return   ERROR ;
}


int rom_class(rom_class)
char  *rom_class;
{
    int   error = ERROR ;
    SEG_DESCRIPTOR *p ;

    /*
       Sets the romable field for the specified class to TRUE permitting
       the output of the segment in the absolute object file.
    */

    p = seg_list ;
    while (p != NULL)   {
       if (strcmp(p->class, rom_class) == 0)    {
          p->romable = TRUE ;
          error = OK ;
       }
       p = p->next ;
    }
    return   error ;
}


int   locate_virtual_segment(vseg, pseg)
unsigned int vseg;
unsigned int *pseg;
{
    SEG_DESCRIPTOR *p ;

    /*
       Finds an initialized segment with the specified virtual segment
       number and returns the corresponding physical segment number.
    */

    p = seg_list ;
```

(continued on next page)

Listing Five *(Listing continued, text begins on page 38.)*

```
    while (p != NULL)    {
       if (p->inited == TRUE && p->vseg == vseg && p->len != 0)    {
          *pseg = p->pseg ;
          return   OK ;
       }
       p = p->next ;
    }
    return   ERROR ;
 }
```

End Listing Five

Listing Six

```
OUTHEX.C

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <string.h>
#include <malloc.h>
#include <dos.h>

#include "loc.h"
#include "externs.h"


void  output_hex_OMF(hex_file, seg_list, entry_point)
int    hex_file ;
SEG_DESCRIPTOR *seg_list ;
unsigned char  *entry_point ;
{
   unsigned int   offset, i, count ;
   unsigned char  *seg_start, *text ;

   SEG_DESCRIPTOR *p ;

   /*
      This function controls the sequencing of the Intel extended hex
      output using the Intel hex output routines.
   */

   /* Run through the segment list and output all ROMable segments */
   p = seg_list ;
   while (p != NULL)    {
      if (p->romable == TRUE)    {

```

```
         /* Allocate enough memory to hold the segment (up to 64K) */
         if ((text = get_mem((unsigned long) p->len)) == NULL)   {
            perror(__FILE__) ;
            exit(1) ;
         }

         /* Locate the position of the segment in the load module file */
         if (lseek(tmp_file, p->position, SEEK_SET) == -1L)   {
            perror(__FILE__) ;
            exit(1) ;
         }

         /* Read in the segment and pad with zero if necessary */
         count = read(tmp_file, text, p->len) ;
         if (count != p->len)    {
            if (count == -1)    {
               perror(__FILE__) ;
               exit(1) ;
            }
            else
               memset(text + count, '\0', p->len - count) ;
         }

         /* Write the segment number out in an address record */
         write_ADDR_record(hex_file, p->pseg) ;

         /* Output the segment as a series of 16 byte data records */
         offset = p->offset ;
         seg_start = text ;
         for (i = 0; i < p->len / 16; i++)    {
            write_DATA_record(hex_file, offset, seg_start, 16) ;
            offset += 16 ;
            seg_start += 16 ;
         }

         /* Handle any remaining data */
         if ((p->len % 16) != 0)
            write_DATA_record(hex_file, offset, seg_start, p->len % 16) ;

         free_mem(text) ;
      }
      p = p->next ;
   }

   /* Write the START and EOF records */
   write_START_record(hex_file, entry_point) ;
   write_EOF_record(hex_file) ;

   return ;
```

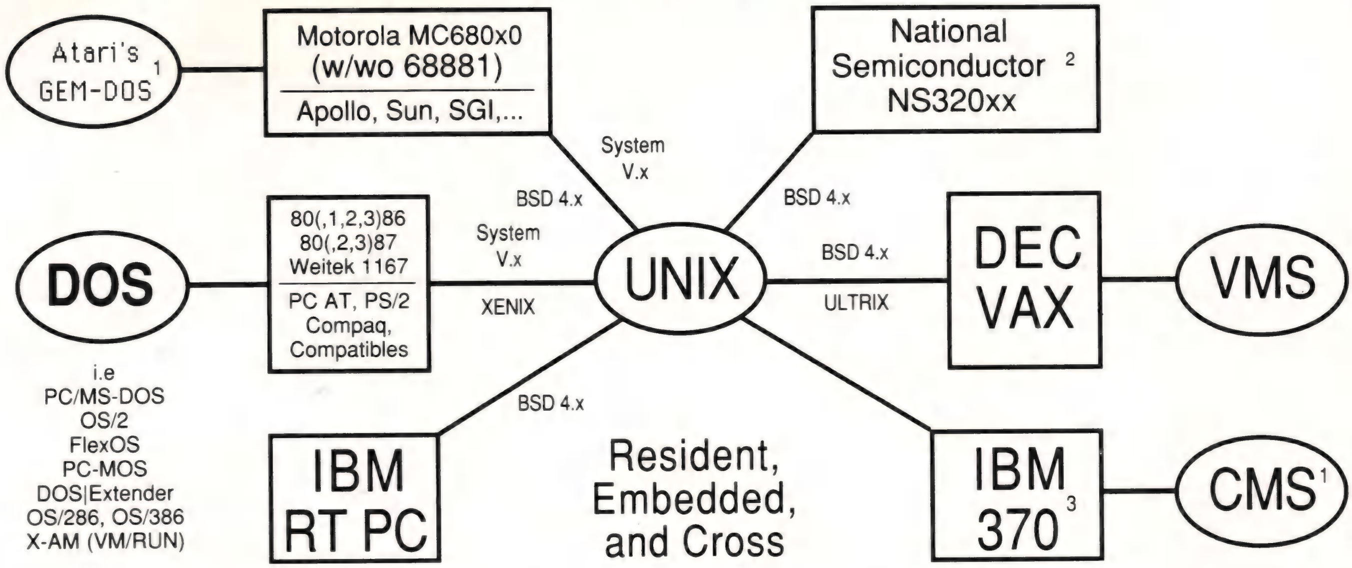
(continued on next page)

Listing Six *(Listing continued, text begins on page 38.)*

```
}


void  write_ADDR_record(file, usba)
int   file ;
unsigned int    usba ;
{
    unsigned char  buf[16], *p ;
    unsigned char  len_field = 2 ;

    /*
        This function writes an Intel extended hex Address record to the
        output file.  Inputs are the file handle and the USBA (segment
        base address).
    */

    p = buf ;
    *p++ = high_byte(usba) ;
    *p++ = low_byte(usba) ;

    output_hex_record(file, ADDR_RECORD, 0, buf, p - buf) ;

    return ;
}


void  write_EOF_record(file)
int   file ;
{
    /*
        This function writes an Intel extended hex EOF record to the
        output file.
    */

    output_hex_record(file, EOF_RECORD, 0, NULL, 0) ;
    return ;
}


void  write_DATA_record(file, offset, text, len)
int   file ;
unsigned int    offset ;
unsigned char   *text ;
unsigned int    len ;
{

    /*
        This function writes an Intel extended hex Data record to the
        specified output file.
    */

    output_hex_record(file, DATA_RECORD, offset, text, len) ;

    return ;
}


void  write_START_record(file, entry)
int   file ;
unsigned char   *entry ;
{
    unsigned char  *buf, *p ;
    unsigned int   count ;

    unsigned char  len_field = 4;
    unsigned int   addr_field = 0 ;
    unsigned char  rec_type = START_RECORD ;

    /*
        This function writes an Intel extended hex Start record to the
        output file.
    */

    /* Allocate some memory to build the data field in */
    if ((p = buf = (unsigned char *) malloc(32)) == NULL)   {
        perror(__FILE__) ;
        exit(1) ;
    }

    /* Store the start address in the data field (segment first) */
    *p++ = high_byte(FP_SEG(entry)) ;
    *p++ = low_byte(FP_SEG(entry)) ;

    /* And then the offset */
    *p++ = high_byte(FP_OFF(entry)) ;
    *p++ = low_byte(FP_OFF(entry)) ;

    /* Output the record */
    output_hex_record(file, START_RECORD, 0, buf, p - buf) ;

    free(buf) ;
    return ;
}


void  output_hex_record(file, type, addr, data, length)
int   file ;
unsigned char   type ;
unsigned int    addr ;
unsigned char   *data ;
unsigned char   length ;
{
    char  *p, *buf ;
    unsigned int   size, count ;
    unsigned char  chksum, digit ;
```

(continued on page 86)

C and Pascal on:



We cut our teeth on UNIX, but have become famous on MS-DOS, which we enhanced with our UNIX-like **DOS Helper** ™ utilities: find (including tar), fgrep, cat, ls, mv, tail, uniq, and wc; and our superior optimizing compilers:

Professional Pascal ™ and **High C** ™ on the PC are now well-respected by organizations such as Ansa, Ashton-Tate, AutoDesk, Boeing (BCS), Daisy Systems Corp., Deloitte Haskins & Sells, Digital Research, GE, IBM, Lifetree, Migent, Multimate, NYU, Silvar-Lisco, Sky Computers, Symantec, Xerox/Ventura, …and *Computer Language* magazine; *Dr. Dobbs' Journal; PC Tech Journal; PC Magazine;* and the *Journal of Pascal, Ada, and Modula-2.*

We supplied the **first** compilers generating **32-bit protected-mode code for the 80386** under MS-DOS (since 11/86). And our newly upgraded MS-DOS real-mode compilers were used by Symantec for their Q&A™ product to exploit the power of the 80386 real-mode instruction set. (**HC v1.4 and PP v2.7 released May 1987.**)

Our **C Validation Suite** will blow your C compiler out of the sea, while our C compiler tracks the emerging ANSI Standard and generates tighter code with far better lint-like feedback help than competing compilers.

And you'll love **Professional Pascal**'s Ada-like packages, true data abstraction, C-like bit manipulation, and much more, along with the tight code that is linkable with **High C**, or other C, object modules (and vice versa).

Our **Translator Writing System (TWS)** goes far beyond LEX and YACC, with fully automatic error recovery…

All uniformly implemented on UNIX, VMS, CMS, MS-DOS, FlexOS, …

***Professional developers** in need of **industrial-strength tools** contact:*

MetaWare Incorporated
903 Pacific Avenue, Suite 201
Santa Cruz, CA 95060-4429

(408) 429-6382

Telex: 493-0879 (META UI)

Meta Ware™ INCORPORATED

PC Tech Journal's conclusion:

The Clear Choice for Large Programming Projects.

Name ____________________

Company ____________________

Address ____________________

City, ST ____________ Zip ________

Phone (____) ____________________

Circle what interests you:

Product: PP HC TWS Helper (DOS only)

Platform: V.x 4.x DOS FlexOS VMS CMS
Sun Apollo Atari VAX 370
8086-family 80386 680x0 32032

Footnotes: 1. Atari, CMS versions available 10/87. 2. NS320xx version by special order. 3. UNIX not yet available on 370.

CIRCLE 158 ON READER SERVICE CARD

Listing Six *(Listing continued, text begins on page 38.)*

```

    /*
       This function does all of the work of writing an Intel extended
       hex output record.  The inputs to this routine are:
          file     output file handle
          type     record type
          addr     address field value
          data     data field contents
          length   size of the data field
    */

    /* Allocate some memory to build the output record in */
    if ((p = buf = malloc(550)) == NULL)    {
       perror(__FILE__);
       exit(1) ;
    }

    /* Build the prefix for the data field */
    p += sprintf(p, ":%02X%02X%02X%02X", length, high_byte(addr), \
       low_byte(addr), type) ;

    /* Compute the checksum on the prefix */
    chksum = length + high_byte(addr) + low_byte(addr) + type ;

    /* Build the data field byte by byte */
    while (length--)    {
       digit = (*data >> 4) & 0x0f ;
       *p++ = (digit > 9) ? digit + 0x37 : digit + '0' ;
       digit = *data & 0x0f ;
       *p++ = (digit > 9) ? digit + 0x37 : digit + '0' ;
       chksum += *data++ ;
    }

    /* Compute the complement of the checksum and output */
    chksum = ~chksum + 1 ;
    p += sprintf(p, "%02X\r\n",chksum) ;

    /* Compute the size of the output record and output */
    size = p - buf ;
    count = write(file, buf, size) ;
    if (count != size)    {
       perror(__FILE__) ;
       exit(1) ;
    }

    free(buf) ;
    return ;
}
```

End Listing Six

Listing Seven

PRINTLOC.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "loc.h"
#include "externs.h"


int    print_statistics(map_filename, stat_filename, command_line, \
         exename, output_file, configname, entry_point)
char   *map_filename ;
char   *stat_filename ;
char   *command_line ;
char   *exename ;
char   *output_file ;
char   *configname ;
unsigned char   *entry_point ;
{
    unsigned long  temp ;
    long  ltime ;
    int    i ;
    SEG_DESCRIPTOR *p ;
    SYMBOL_LIST *q ;

    /*
       This function generates the locate map file.  The locate map
       file contains the segment information with the physical
       segment addresses.
    */

    fprintf(print_file, "MS-DOS Locate Utility Version 1.0\n\n") ;
    fprintf(print_file, "Input File: %s\n", exename) ;
    fprintf(print_file, "Output File: %s\n", output_file) ;
    fprintf(print_file, "Configuration File: %s\n", configname) ;
    fprintf(print_file, "Invoked by: %s\n", command_line) ;

    time(&ltime) ;
    fprintf(print_file, "Date/Time: %s\n", ctime(&ltime)) ;

    /* Display the located segment information */
    fprintf(print_file, "Segment Information\n");
    fprintf(print_file, "%-16s%-16s%-12s%-12s\n", "Name", "Class",
               "Address", "Length");

    p = seg_list ;
    while (p != NULL)    {
       temp = (unsigned long) p->pseg ;
       temp = temp * 16 + p->offset ;
       fprintf(print_file, "%-16s%-16s%05lXH%10.04XH\n", p->name,
                 p->class, temp, p->len) ;
       p = p->next ;
    }
```

```
    /* Display the symbol information */
    fprintf(print_file, "\n\nPublic Symbols\n");
    i = 0 ;
    p = seg_list ;
    while (p != NULL)    {
       q = p->symbol_list ;
       while (q != NULL)    {
          fprintf(print_file, "%04X:%04X  %-16s%s", p->pseg,
                   q->value, q->name, i++ ? "\n" : "\t\t") ;
          i %= 2 ;
          q = q->next ;
       }
       p = p->next ;
    }

    fprintf(print_file, "%sEntry Point - %p\n", (i == 1) ? "\n\n" : "\n",
             entry_point) ;

    return ;
 }
```

End Listing Seven

Listing Eight

READCFG.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "loc.h"
#include "externs.h"

int   process_class_keyword(), process_order_keyword(), process_null() ;
int   process_rom_keyword(), process_dup_keyword(), process_comment() ;


static   struct   CFG_COMMANDS    {
            char   *cmd ;
            int    (*command) () ;
      }  cfg_cmds[] = {
            "CLASS",      process_class_keyword,
            "ORDER",      process_order_keyword,
            "ROM",        process_rom_keyword,
            "DUP",        process_dup_keyword,
            ";",          process_comment
         } ;


int   process_locate_file(seg_list)
```

(continued on next page)

Listing Eight *(Listing continued, text begins on page 38.)*

```
SEG_DESCRIPTOR *seg_list ;
{
   int   i, error = OK;
   char  *tok, *buf ;

   /*
      This function reads the configuration file and performs the parsing
      and control transfer to routines which perform the desired action.
   */

   /* Allocate some memory for the line buffer */
   if ((buf = malloc(256)) == NULL)    {
      perror(__FILE__) ;
      exit(1) ;
   }

   /* Read and categorize a token from the configuration file */
   while (fgets(buf, 256, config_file) != NULL)    {
      /* Extract the first token (read the next line if none is found */
      if ((tok = strtok(buf, " \t\n")) == NULL)
         continue ;

      for (i = 0; i < dim(cfg_cmds); i++)    {
         if (stricmp(cfg_cmds[i].cmd, tok) == 0)    {
            error = (*cfg_cmds[i].command) () ;
            break ;
         }
      }

      if (i == dim(cfg_cmds))    {
         fprintf(stderr, "Illegal input - <%s>\n", tok) ;
         exit(1) ;
      }
   }
   free(buf) ;
   return   error ;
}


int   process_comment()
{
   return   OK ;
}


int   process_class_keyword()
{
   char  *tok, name[32], *p ;
   unsigned int   seg ;

   /*
      This function parses the remainder of the CLASS directive.
   */

   /* Read the class name */
   strcpy(name, strupr(strtok(NULL, " \t\n="))) ;

   /* Verify that an equal sign is present */
   if (strcmp((tok = strtok(NULL, " \t\n")), "=") != 0)    {
      fprintf(stderr, "\"=\" expected instead found <%s>\n", tok) ;
      return   ERROR ;
   }

   /* Read the segment number for the class */
   tok = strtok(NULL, " \t\n") ;
   seg = (unsigned int) strtol(tok, &p, 0) ;
   if (*p)    {
      fprintf(stderr, "Unrecognized token <%s>\n", p) ;
      return   ERROR ;
   }

   /* Assign the physical segment number to the specified class */
   if (assign_physical_segment(name, seg) == ERROR)    {
      fprintf(stderr, "Undefined class <%s>\n", name);
      return   ERROR;
   }

   return   OK ;
}


int   process_order_keyword()
{
   char  *tok, pclass[32], class[32] ;
   unsigned int   next_seg ;
   BOOLEAN   found = FALSE ;


   /*
      This function processes the ORDER directive.
   */

   /* Read the leading class name from the command */
   strcpy(pclass, strupr((tok = strtok(NULL, " \t\n")))) ;

   /* Process the remaining class names in the command */
   while ((tok = strtok(NULL, " \t\n")) != NULL)    {
      if (*tok == ';')
         break ;

      found = TRUE ;
      strcpy(class, strupr(tok)) ;

      /* Compute the segment address for this class to be made
         contiguous with the previous class */
      if (get_next_segment(pclass, class, &next_seg) == ERROR)    {
```

(continued on page 90)

DOS LOCATE UTILITY

Listing Eight (*Listing continued, text begins on page 38.*)

```
            fprintf(stderr, "Undefined class <%s>\n", pclass) ;
            return    ERROR ;
         }

         /* Assign the computed segment number to the class */
         if (assign_physical_segment(class, next_seg) == ERROR)    {
            fprintf(stderr, "Undefined class <%s>\n", class) ;
            return    ERROR ;
         }

         /* Setup to process the next class */
         strcpy(pclass, class) ;
      }

      return    (found == FALSE) ? ERROR : OK ;
}


int    process_dup_keyword()
{
      char  *tok, old_class[32], new_class[32] ;


      /*
         This function is responsible for processing the DUP directive.
      */

      /* Read the existing class name */
      strcpy(old_class, strupr((tok = strtok(NULL, " \t\n")))) ;

      /* Read the name of the class to be created */
      strcpy(new_class, strupr((tok = strtok(NULL, " \t\n")))) ;

      /* Duplicate the existing class */
      if (dup_class(old_class, new_class) == ERROR)    {
         fprintf(stderr, "Undefined class <%s>\n", old_class);
         return    ERROR ;
      }

      return    OK ;
}


int    process_rom_keyword()
{
      char  *tok, class[32] ;

      /*
         This function processes the ROM keyword and marks all specified
         classes as ROMable.
      */

      /* Read all of the tokens on the line */
      while ((tok = strtok(NULL, " \t\n")) != NULL)    {
         if (*tok == ';')
            break ;

         strcpy(class, strupr(tok)) ;

         if (rom_class(class) == ERROR)    {
            fprintf(stderr, "Undefined class <%s>\n", class) ;
            return    ERROR ;
         }
      }

      return    OK ;
}
```

End Listing Eight

Listing Nine

READMAP.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>

#include "loc.h"
#include "externs.h"

#define  BUFSIZE   256


SEG_DESCRIPTOR *build_seg_list()
{
      int count ;
      unsigned long  start_seg, end_seg, length;
      char  seg_name[32], class[32] ;
      char  *buf ;

      SEG_DESCRIPTOR *p, *previous, *list_start, *class_start ;

      /*
         This function is responsible for the processing of the link map.
         The link map is read and the segment information such as segment
         name, segment length and class name are recorded.
      */

      /* Seek to the beginning of the file */
      if (fseek(map_file, 0L, SEEK_SET) != 0)    {
         perror(__FILE__) ;
         exit(1) ;
      }
```

```
    /* Allocate some memory for the line buffer */
    if ((buf = malloc(BUFSIZE)) == NULL)    {
        perror(__FILE__) ;
        exit(1) ;
    }

    /* Search thru the file looking for the start of the segment informati/
    while (1)    {
        if (fgets(buf, BUFSIZE, map_file) == NULL)    {
            fprintf(stderr, "Unable to find the segment list in %s\n", map_f;
            exit(1) ;
        }

        if (strstr(strupr(buf), "START") != NULL)
            break ;
    }

    /* Scan to the start of the first segment record */
    while (fgets(buf, BUFSIZE, map_file) != NULL)    {
        count = sscanf(buf, " %lxH %lxH %lxH %s %s", &start_seg, &end_seg, ;
        if (count == 5)
            break ;
    }

    /* Check if EOF was detected and an error message should be printed */
    if (feof(map_file))    {
        fprintf(stderr, "Unable to find the segment list in %s\n", map_fnam;
        exit(1) ;
    }

    /* Begin processing the list of segments */
    p = previous = NULL ;
    while (count == 5)    {
        /* Allocate some memory to hold the data structure */
        if ((p = (SEG_DESCRIPTOR *) malloc(sizeof (*p))) == NULL)    {
            perror(__FILE__) ;
            exit(1) ;
        }

        if (previous == NULL)
            list_start = p ;
        else
            previous->next = p ;

        strcpy(p->name, strupr(seg_name)) ;
        strcpy(p->class, strupr(class)) ;
        p->vseg = (unsigned int) start_seg / 16 ;
        p->offset = (unsigned int) start_seg % 16 ;
        p->len = (unsigned int) length ;
        p->position = start_seg ;
        p->inited = FALSE ;
        p->romable = FALSE ;
        p->symbols = 0 ;
        p->symbol_list = NULL ;
        p->next = NULL ;

        /* Check if the class name has changed and reset the offset */
        if (strcmp(p->class, class_start->class) != 0)    {
            p->pseg = 0 ;
            class_start = p ;
        }
        else
            p->pseg = p->vseg - class_start->vseg ;

        previous = p;

        /* Read the next line of segment information */
        fgets(buf, BUFSIZE, map_file) ;
        count = sscanf(buf, " %lxH %lxH %lxH %s %s", &start_seg, &end_seg, ;
    }

    free(buf) ;
    return (list_start) ;
}
```

End Listing Nine

Listing Ten

SIEVE.C

```
/*
    Sieve Benchmark - ROM Version

    Copyright (C) Recycled Software 1987.  All rights reserved.

    Executes 100 Iterations of the sieve algorithm for microprocessor
    benchmarking purposes.

*/


#define  TRUE     1
#define  FALSE    0
#define  SIZE     8190

char  flags[SIZE + 1] ;

main()
{
    int   i, prime, k, count, iter ;

```

(continued on next page)

Listing Ten *(Listing continued, text begins on page 38.)*

```
   for (iter = 1; iter <= 100; iter++)    {
      count = 0 ;
      for (i = 0; i <= SIZE; i++)
         flags[i] = TRUE ;

      for (i = 0; i <= SIZE; i++)    {
         if (flags[i])    {
            prime = i + i + 3 ;
            for (k = i + prime; k <= SIZE; k += prime)
               flags[k] = FALSE ;

            count++ ;
         }
      }
   }
}
```

End Listing Ten

Listing Eleven

STABLE.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>

#include "loc.h"
#include "externs.h"

char  *errstr = "Unable to locate global symbol \"%s\"\n" ;


void  read_symbol_table(seg_list)
SEG_DESCRIPTOR *seg_list ;
{
   char  buf[128] ;
   int   count, found ;
   unsigned int   vseg, off;
   char  symbol[32], attrb[10] ;

   SEG_DESCRIPTOR *p ;
   SYMBOL_LIST *q ;

   /*
      This function reads the linker map file and extracts the
      symbol information.
```

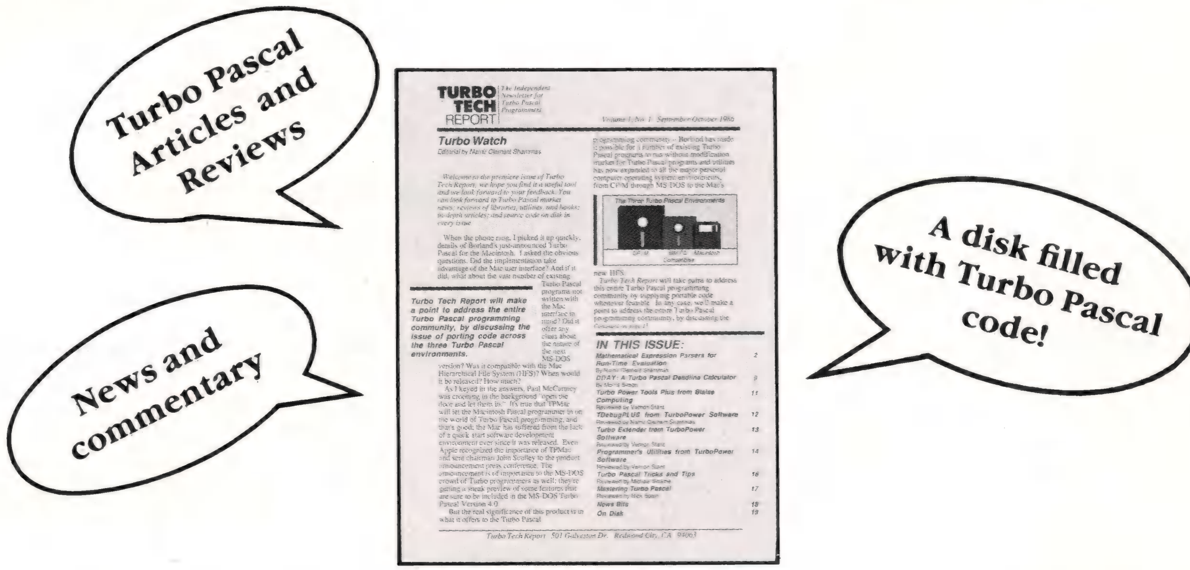
(continued on page 94)

Listing Eleven *(Listing continued, text begins on page 38.)*

```
     */

     /* Seek to the beginning of the file */
     if (fseek(map_file, 0L, SEEK_SET) != 0)   {
        perror(__FILE__) ;
        exit(1) ;
     }

     /* Search thru the file for the symbol tables */
     while (1)   {
        if (fgets(buf, sizeof(buf), map_file) == NULL)
           return ;

        if (strstr(strupr(buf), "ADDRESS") != NULL)
           break ;
     }

     /* Read each of the symbol entries */
     while (1)   {
        count = fscanf(map_file, " %4x:%4x%5c %s", &vseg, &off, attrb, symb;
        if (count != 4)
           break ;
        else    {
           p = seg_list ;
           found = FALSE ;
           while (p != NULL)    {
              if ((p->vseg != vseg) || (p->len == 0))    {
                 p = p->next ;
                 continue ;
              }

              q = (SYMBOL_LIST *) malloc(sizeof(*q)) ;
              strcpy(q->name, symbol) ;
              q->value = off ;
              q->type = 1 ;
              q->next = p->symbol_list ;
              p->symbol_list = q ;
              p->symbols++ ;
              found = TRUE ;
              break ;
           }

           if (found == FALSE)
              fprintf(stderr, errstr, symbol) ;
        }
     }

     return ;
  }
```

End Listing Eleven
(Listings will continue next month)

Turbo Tech Report Speaks Your Language.



The newsletter/disk publication for Turbo Pascal® users

Are you looking for powerful utilities written in Turbo Pascal that you can use to develop software or incorporate into your programs? Are you interested in improving and expanding your Turbo Pascal programming skills?

Then you deserve a subscription to *Turbo Tech Report*, the bimonthly newsletter/disk publication from the publishers of *Dr. Dobb's Journal* and *Micro/Systems Journal*. Each issue delivers more than 250K of Turbo Pascal source code programs on disk, and 24+ pages of articles, Turbo Pascal software and book reviews, and analysis and commentary. It's the only publication delivering such focused technical articles with code on disk. Each valuable issue contains:

• ***Articles*** on topics like speedy 3D graphics, mathematical expression parsers, creating global gotos, memory resident and AI applications and more—all written by Turbo experts.

• ***Reviews*** of the latest Turbo Pascal software programs from companies like Borland International, Blaise Computing, Media Cybernetics, Nostradamus, and more!

• ***News and commentary*** detailing the latest products and developments in the Turbo Pascal programming community.

• ***A disk filled with Turbo Pascal code!*** You'll get the Turbo Pascal utilities and routines discussed in the newsletter's articles, as well as applications developed by Turbo users from around the world. You'll receive programs that make labels, generate menus, provide faster screen access, transfer files, and more!

If you're an expert Turbo Pascal programmer or a novice interested in expanding your Turbo skills, you need a publication that speaks your language: *Turbo Tech Report*. Subscribe today at the special price of just $99—that's 33% off the regular price of $150. To order by credit card, call toll-free 1-800-533-4372. Or mail the coupon with your payment to *Turbo Tech Report*, 501 Galveston Drive, Redwood City, CA 94063.

—**Yes!** I want a one-year subscription to *Turbo Tech Report* (6 issues with 6 disks) for $99.

Format: ☐ PC/MS-DOS ☐ Macintosh

CP/M: ☐ Kaypro ☐ Osborne ☐ Apple

PAYMENT MUST ACCOMPANY ALL ORDERS

☐ Check/money order enclosed.

☐ Charge my: ___ VISA ___ M/C ___ AmExp.

Card # ________________ Exp. ________

Signature ________________

Name ________________

Address ________________

City ________ State ______ Zip ______

Orders outside U.S.: add $30.

Ask for a FREE REPORT: "Interviews with Authors & Users:

How Can These Products Raise Productivity and Help You Write Better Programs?" from

THE PROGRAMMER'S SHOP

EXIM TOOLKIT:

ESSENTIAL BASIC PROGRAMMING TOOLS

A **must** for all BASIC programmers who use Microsoft's QuickBASIC or IBM's PC BASIC compilers, our feature-rich, easy-to-use **EXIM ToolKit** provides dozens of useful routines unavailable in BASIC or any other single software library sold today!

Here's just some of what the **EXIM ToolKit** can do:

With our **Screen Management** module, you get features like four-way scrolling, windowing and data entry/display in screen forms. You can also write to screen many times faster than with BASIC and develop software to copy areas of the screen for cut and paste functions.

The multi-user **Data Base Management** module lets multiple users access the same data files across a local area network and still maintain data integrity. Users can build, add and delete indexes; reformat files and much more. Now BASIC users can control data as never before.

The **Network Interface modules** make it possible for BASIC programs to issue NETBIOS commands.

BASIC limits you to 64K of RAM for strings. **EXIM ToolKit's Memory Manager** lets you access up to **256K additional RAM** for strings. And, we offer free technical support, 7 days a week.

Order your **EXIM ToolKit** today. It's only $99.95, ($199.95 with Network Interface Modules) plus $5.00 shipping and handling. (No licensing or royalties required.) If within 30 days, you are not completely satisfied with the **EXIM ToolKit**, return it to your dealer for a complete refund.

The **EXIM ToolKit** is so comprehensive, we're convinced it's the only BASIC Software Library you'll ever need.

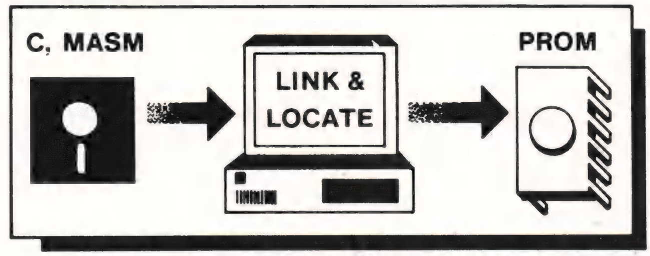
PS: $ 45

EXIM

EXIM SERVICES OF N.A. INC.
P.O. Box 5417
Clinton, New Jersey 08809
(201) 735-7640

CIRCLE 172 ON READER SERVICE CARD

FIRMWARE DEVELOPMENT



LINK & LOCATE enables PC users to produce ROM-based firmware for 8086/87/186 from object files generated by popular C compilers, such as from Microsoft, Lattice and Borland's Turbo C, and MASM from Microsoft, Provides full control of segment placement anywhere in memory. Supports output of Intel HEX file for PROM programmers, Intel OMF absolute object file for symbolic debuggers and in-circuit emulators. Includes Intel compatible linker, locator, librarian, hex formatter and cross reference generator. $350.

NEW! Includes utility to support PC based PROM programmers.

SYSTEMS & SOFTWARE INC. **PS: $329**

3303 Harbor Blvd., C11, Costa Mesa, CA 92626
Phone (714) 241-8650 FAX (714) 241-0377 TWX 910-695-0125

CIRCLE 173 ON READER SERVICE CARD

Now you see it; Now you don't!

TSR's made easy with /*resident_C*/

Finally! You write your C program and we make it resident. No strings attached. Want to process interrupts also? No problem. **/*resident_C*/** is a set of C functions that enable you to process interrupts and/or make your programs "terminate stay resident" like Sidekick.

We've done the research, testing, and grunt work to make your TSR programs safe, compatible and easy. Be aware of the "ifs, ands and buts" when writing interrupt handling software. Only **/*resident_C*/** has all that you need to write reliable interrupt handlers without the worries.

We don't ask you to trust us either. Our manual explains what we are doing very clearly and our documented source code is available. In addition, our working demos give you clear examples of resident programs, interrupt handlers and resident libraries. There is no other product that can do what **/*resident_C*/** can do for you.

/* resident_C */ is the perfect complement to the other library products available from Essential Software.

ESI

List: $99 / with Source add $99
PS: $79 / with Source add $79

CIRCLE 174 ON READER SERVICE CARD

Call or Circle Reader Service Number for Your FREE Study.

HOURS: 8:30 A.M.-8:00 P.M. E.S.T.
5-D Pond Park Road, Hingham, MA 02043
Mass: 800-442-8070 or 617-740-2510

800-421-8006

THE PROGRAMMER'S SHOP
Your complete source for software, services and answers

Listing One *(Text begins on page 48.)*

```
Listing 1

;         ISQRT32.ASM - 32 bit integer test program for no 8087
;         By Ray Mariella, March 87
           page    ,96
;
crlf      macro
          mov     dl,13
          call    char_out
          mov     dl,10
          call    char_out
          endm
;
time_print macro byte_var, byte_string
          local plenty
          mov     dl,byte_string              ;output a colon or period
          call    char_out
          cmp     byte_var,9
          ja      plenty
          mov     dl,'0'                      ;space holder if var<10
          call    char_out
plenty:   mov     al,byte_var                 ;minutes, secs, or hnds
          xor     ah,ah
          call    dec_out
          endm
;
;
data      segment  word public 'DATA'
base              dw  10                        ;base for dec_out
uper              db  ?                         ;for time_print routine
secs              db  ?
hnds              db  ?
announc           db  ' 60000   32 bit square roots ',13,10,'$'
data      ends
;
;
stack     segment   stack
                  dw  64  dup(?)
stack     ends
;
;
code      segment  word public 'code'
          assume cs:code, ds:data, ss:stack
;
sqrt:     mov     ax,data
          mov     ds,ax
crlf
          mov     dx,offset announc
          mov     ah,9                      ;print string function
          int     21h                       ;DOS interrrupt
          mov     dl,13
          call    char_out
;
herald:   crlf
;
rolling:  xor     di,di                     ;upper 16
          mov     si,32767                  ;lower 16
;
goodies:  call    update
;
;      square root procedure of DI:SI via 8086,
;
start:    mov     bx,1                      ;initial value for infimum
          mov     dx,di                     ;initial supremum, upper 16
          mov     ax,si                     ;initial supremum, lower 16
                                            ;
biggest:  or      dx,dx                     ;test if upper 16 =0 yet
          jz      words                     ;if yes, we don't need upper 16 now
          rcr     dx,1                      ;supr. upper 16/2
          rcr     ax,1                      ;supr. lower 16/2 + carry from upper
          shl     bx,1                      ;infim.*2
          jmp     short biggest
                                            ;now infim. and supr. are 16 bits
words:    or      bx,bx                     ;if BX was made 0, correct it!
          jnz     checkem                   ;if not, O.K. to continue
          mov     bx,0ffffh                 ;correction for the largest 32 bitters
checkem:  mov     dx,ax                     ;supr. in ax,dx
          mov     cx,bx                     ;infim. in bx,cx
                                            ;
logit:    shl     cx,1                      ;infimum*2
          jc      average                   ;necessary for large integers
          cmp     cx,dx                     ;infimum*2 > supremum?
          jae     average                   ;if so, ready to average
          shr     dx,1                      ;if not, supr/2
          mov     ax,dx                     ;store latest values
          mov     bx,cx
          jmp     short logit
                                            ;ready for averaging
average:
          add     bx,ax                     ;(infim.+ supr.)
          rcr     bx,1                      ;average value for first guess
Newton:
          REPT 2
          mov     ax,si                 ;lower 16 of target in ax,
          mov     dx,di                 ;upper 16 of in dx, for division
;         cmp     bx,dx                 ;this is for near FFFE:0000 and up
;         je      cont                  ;but not needed for FFFD:0000 and less
          div     bx                    ;N/(g1) in AX, now get g2
          add     bx,ax                 ;Newton's method   g2 = (g1 +N/g1)/2
          rcr     bx,1                  ;bx now has g2
          endm
;
cont:     inc     di
          cmp     di, 60000
          ja      quit
          jmp     start
;
quit:     call    update
```

(continued on page 100)

Listing One *(Listing continued, text begins on page 48.)*

```
            xor     al,al
            mov     ah,4Ch
            int     21h
;
;           output a hex word in decimal
;
;           CX,AX,DX destroyed
;
dec_out     proc    near
            xor     cx,cx
another:    inc     cx
            xor     dx,dx
            div     base                        ;base is 10 decimal!
            push    dx                          ;remainder is less sig digits
            or      ax,ax                       ;is the quotient zero?
            jnz     another                     ;if not, more number to convert
print_dig:
            pop     dx                          ;retrive digit from stack
            add     dl,'0'                      ;ascii offset
            call    char_out
            loop    print_dig                   ;do all of the digits
            ret
dec_out     endp
;
;           output a single character
;
char_out    proc    near
            mov     ah,2                        ;output char function
            int     21h                         ;do it
            ret
char_out    endp
;
;
update      proc   near
            mov     ah,2ch                      ;get dos time
            int     21h                         ;hour in ch, mins in cl,secs in dh
            mov     uper,cl
            mov     secs,dh
            mov     hnds,dl
            mov     al,ch
            xor     ah,ah
            call    dec_out
;
            time_print uper,':'
;
            time_print secs,':'
;
            time_print hnds,'.'
;
            crlf
            ret
update      endp
;
code        ends
            end     sqrt
```

End Listing One

Listing Two

```
Listing 2

;          R32COMP.ASM - 32 bit sqr , compares CPU, NDP
;          By Ray Mariella, 30 March 87 - increments the upper 16 bits
;                                         0000:7FFF to FFFF:7FFF
;          requires 8087 or 80287
            page    ,96
.8087
;
crlf        macro
            mov     dl,13
            call    char_out
            mov     dl,10
            call    char_out
            endm
;
time_print macro byte_var, byte_string
            local plenty
            mov     dl,byte_string          ;output a colon or period
            call    char_out
            cmp     byte_var,9
            ja      plenty
            mov     dl,'0'                  ;space holder if var<10
            call    char_out
plenty:     mov     al,byte_var             ;minutes, secs, or hnds
            xor     ah,ah
            call    dec_out
            endm
;
;
data        segment  public 'DATA'
            even
base                dw  10                     ;base to print the numbers in
BIGGUN              dq  ?
rootp               dd  ?
uper                db  ?
secs                db  ?
hnds                db  ?
announc             db  ' 65535 increments of upper 16, CPU then 8087 ',13,10,'$'
data        ends
;
;
stack       segment   stack
                    dw  64  dup(?)
stack       ends
;
;
code        segment  public 'CODE'
           assume cs:code, ds:data, ss:stack
;
sqrt:       push    bp
```

```
         mov     ax,data
         mov     ds,ax
         mov     bp,offset biggun
crlf
         mov     dx,offset announc
         mov     ah,9                    ;print string function
         int     21h                     ;DOS interrrupt
;
         mov     si,32767
         xor     di,di
goodies: call    update
;
;        square root procedure via 8086, DI:SI
;
CPU:     mov     bx,1                    ;guess1
         mov     dx,di                   ;guess2
         mov     ax,si
biggest: or      dx,dx
         jz      words
         rcr     dx,1                    ;upper 16/2
         rcr     ax,1                    ;lower 16/2 + carry from upper
         shl     bx,1                    ;guess1*2
         jmp     short biggest
                                         ;next is for guess1 and guess2 16 bits
words:   or      bx,bx
         jnz     checkem                 ;if all 32 were used, CF is set
         mov     bx,65535                ;in case all 32 bits were used
;
checkem: mov     dx,ax                   ;guess2 ax,dx
         mov     cx,bx                   ;guess1 bx,cx
;
logit:   shl     cx,1                    ;guess1
         jc      average                 ;necessary for very large integers
         cmp     cx,dx                   ;larger than guess2?
         jae     average                 ;if not, guess2/2
         shr     dx,1
         mov     ax,dx
         mov     bx,cx
         jmp     short logit
                                         ;ready for averaging
average:
         add     bx,ax
         rcr     bx,1                    ;average value
Newton:
         REPT    2
         mov     ax,si               ;lower 16
         mov     dx,di               ;prepare for division, upper 16 in dx
         cmp     bx,dx                   ;needed for really BIG ints
         je      quit                    ; "
         div     bx                  ;ax still has target, bx first guess
         add     bx,ax               ;newton
         rcr     bx,1
         endm
;

quit:    inc     di
         jz      done
         jmp     CPU
;
done:    call    update
;
;        square root via 8087
;  if you need roots of 7FFF:FFFF and less, BIGGUN can be 32 bits,
;  and ROOTP can be a 16 bits.  The extra length is needed here because
;  the 8087 does not expect unsigned integers.
;
         xor     di,di                   ;8087 loads from memory,
         mov     ds:[bp],si              ;not regs directly
NDP:     mov     ds:[bp+2],di
        fild    biggun                  ;put integer into 8087 stack
        fsqrt
        fistp   rootp                   ;store to memory, too
        fwait
;
; we now have an 8087 square root -rootp
;
         inc     di
         jnz     NDP

         call    update
         pop     bp
         xor     al,al
         mov     ah,4Ch
         int     21h
;
;
;        output a hex word in decimal
;
;        CX,AX,DX destroyed
;
dec_out  proc    near
         xor     cx,cx
another: inc     cx
         xor     dx,dx
         div     base                    ;base is 10 decimal!
         push    dx                      ;remainder is less sig digits
         or      ax,ax                   ;is the quotient zero?
         jnz     another                 ;if not, more number to convert
print_dig:
         pop     dx                      ;retrive digit from stack
         add     dl,'0'                  ;ascii offset
         call    char_out
         loop    print_dig               ;do all of the digits
         ret
dec_out  endp
;
;        output a single character
;
```

(continued on next page)

Listing Two *(Listing continued, text begins on page 48.)*

```
char_out  proc   near
          mov    ah,2                    ;output char function
          int    21h                     ;do it
          ret
char_out  endp
;
;
update    proc  near
          mov    ah,2ch                  ;get dos time
          int    21h                     ;hour in ch, mins in cl,secs in dh
          mov    uper,cl
          mov    secs,dh
          mov    hnds,dl
          mov    al,ch
          xor    ah,ah
          call   dec_out
;
          time_print uper,':'
;
          time_print secs,':'
;
          time_print hnds,'.'
;
          crlf
          ret
update    endp
;
code      ends
          end    sqrt
```

End Listing Two

Listing Three

```
Listing 3

;         R32fail.ASM - 32 bit integer test program REPT macros
;         looks for the first time that Newton fails 0,-1
;         By Ray Mariella, April 87
           page    ,96
.8087
;
crlf      macro
          mov    dl,13
          call   char_out
          mov    dl,10
          call   char_out
          endm
;
time_print macro byte_var, byte_string
          local plenty
          mov    dl,byte_string              ;output a colon or period
          call   char_out
          cmp    byte_var,9
          ja     plenty
          mov    dl,'0'                      ;space holder if var<10
          call   char_out
plenty:   mov    al,byte_var                 ;minutes, secs, or hnds
          xor    ah,ah
          call   dec_out
          endm
;
;
data      segment  word public 'DATA'
base             dw  10                      ;base to print the numbers in
BIGGUN           dq  ?
rootp            dd  ?
contwd           dw  ?
uper             db  ?
secs             db  ?
hnds             db  ?
announc          db  'incr. lower 16 from 1 ',13,10,'$'
intermed         db  ' passed 65535   ',13,10,'$'
data      ends
;
;
stack     segment   stack
                 dw  64  dup(?)
stack     ends
;
;
code      segment  word public 'code'
          assume cs:code, ds:data, ss:stack
;
sqrt:     push   bp
          mov    ax,data
          mov    ds,ax
          mov    bp,offset biggun
          fclex                          ;clear 8087 exceptions, if any
          fstcw  contwd                  ;get control word
          and    contwd,1111001111111111B ;round to nearest
          fldcw  contwd                  ;load changed control word
          crlf
          mov    dx,offset announc
          mov    ah,9                     ;print string function
          int    21h                      ;DOS interrrupt
          mov    dl,13
          call   char_out
;
herald:   crlf
;
rolling:  xor    bx,bx
          xor    ax,ax
          mov    si,1                    ;lower 16, will vary
          mov    ds:[bp],si              ;BIGGUN lower 16
          mov    di,0                    ;upper 16
          mov    ds:[bp+2],di
;
;
```

```
goodies: call   update
         crlf
start:   FILD BIGGUN
;
;       square root procedure via 8086/V30
;
;
         mov    bx,1                    ;guess1
         mov    dx,di                   ;guess2
         mov    ax,si
;
         REPT 8
         or     dx,dx
         jz     halfway
         rcr    dx,1                    ;upper 16/2
         rcr    ax,1                    ;lower 16/2 + carry from upper
         shl    bx,1                    ;guess1*2
         endm
         jmp    short rest
halfway: jmp    short words
;
rest:
         REPT 8
         or     dx,dx
         jz     words
         rcr    dx,1                    ;upper 16/2
         rcr    ax,1                    ;lower 16/2 + carry from upper
         shl    bx,1                    ;guess1*2
         endm                           ;next is for guess1 and guess2 16 bits
words:   FSQRT
         or     bx,bx
         jnz    checkem                 ;if all 32 were used, CF is set
         mov    bx,65535                ;in case all 32 bits were used
;
checkem: mov    dx,ax                   ;guess2 ax,dx
         mov    cx,bx                   ;guess1 bx,cx
;
logit:
         REPT 8
         shl    cx,1                    ;guess1
         jc     average                 ;necessary for 2000:4000 and up
         cmp    cx,dx                   ;larger than guess2?
         jae    average
         shr    dx,1                    ;if not, guess2/2
         mov    ax,dx
         mov    bx,cx
         endm
                                        ;ready for averaging
average: FISTP  rootp
         add    bx,ax
         rcr    bx,1                    ;average value
Newton:
         REPT   2
         mov    ax,si           ;lower 16
         mov    dx,di           ;prepare for division, upper 16 in dx
         cmp    bx,dx           ;for FFFE:0000 and up
         je     quit
         div    bx              ;ax still has target, bx first guess
         add    bx,ax           ;newton
         rcr    bx,1
         endm
         FWAIT
done:    mov    ax,bx                           ;ax, and bx have approx. root
         mov    dx,word ptr ds:[bp+8]           ;bp+8 is rootp, 8087 root
         xor    ax,dx                           ;see if rootp agrees
         jz     cont
         cmp    bx,dx
         ja     quit
;
belo:    inc    bx
         cmp    bx,dx
         jnz    quit
;
;
cont:    inc    si                      ;lower 16
         jnz    notyet
         inc    di
         mov    ax,di
         call   dec_out
         mov    dx,offset intermed
         mov    ah,9
         int    21h
         crlf
notyet:  mov    ds:[bp+2],di            ;biggun upper 16
         mov    ds:[bp],si              ;biggun lower 16
         jmp    start
;
quit:    call   update
         mov    ax,di
         call   dec_out
         mov    dl,':'
         call   char_out
         mov    ax,si
         call   dec_out
         crlf
         pop    bp
         xor    al,al
         mov    ah,4Ch
         int    21h
         ret
;
;        output a hex word in decimal
;
;        CX,AX,DX destroyed
;
dec_out  proc   near
         xor    cx,cx
another: inc    cx
```

(continued on next page)

Listing Three *(Listing continued, text begins on page 48.)*

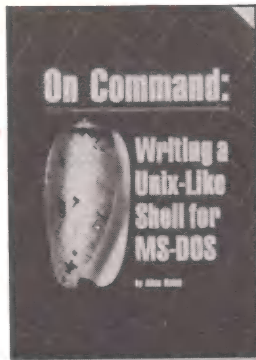
```
            xor     dx,dx
            div     base                    ;base is 10 decimal!
            push    dx                      ;remainder is less sig digits
            or      ax,ax                   ;is the quotient zero?
            jnz     another                 ;if not, more number to convert
print_dig:
            pop     dx                      ;retrive digit from stack
            add     dl,'0'                  ;ascii offset
            call    char_out
            loop    print_dig               ;do all of the digits
            ret
dec_out     endp
;
;           output a single character from dl
;
char_out    proc    near
            mov     ah,2                    ;output char function
            int     21h                     ;do it
            ret
char_out    endp
;
;
update      proc  near
            mov     ah,2ch                  ;get dos time
            int     21h                     ;hour in ch, mins in cl,secs in dh
            mov     uper,cl
            mov     secs,dh
            mov     hnds,dl

            mov     al,ch
            xor     ah,ah
            call    dec_out
;
            time_print uper,':'
;
            time_print secs,':'
;
            time_print hnds,'.'
;
            crlf
            ret
update      endp
;
code        ends
            end     sqrt
```

End Listing Three

Listing Four

```
Listing 4

;           RALL16 - square roots of 1 to 65535
;           By Ray Mariella, Nov 1986
             page    ,96
;
crlf        macro
            mov     dl,13
            call    char_out
            mov     dl,10
            call    char_out
            endm
;
time_print macro byte_var, byte_string
            local plenty
            mov     dl,byte_string          ;output a colon or period
            call    char_out
            cmp     byte_var,9
            ja      plenty
            mov     dl,'0'                  ;space holder if var<10
            call    char_out
plenty:     mov     al,byte_var             ;minutes, secs, or hnds
            xor     ah,ah
            call    dec_out
            endm
;
;
data        segment  word public 'DATA'
            even
base                dw  10                  ;base to print the numbers in
uper                db  ?
secs                db  ?
hnds                db  ?
announc             db  'square roots of 1 - 65535 ',13,10,'$'
data        ends
;
;
stack       segment   stack
                    dw  64  dup(?)
stack       ends
;
;
code        segment  word public 'CODE'
            assume cs:code, ds:data, ss:stack
            even
;
sqrt:       mov     ax,data
            mov     ds,ax
crlf
            mov     dx,offset announc
            mov     ah,9                    ;print string function
            int     21h                     ;DOS interrrupt
            mov     dl,13
            call    char_out
;
```

MS-DOS Enhancements and Unix-Like Features for MS-DOS



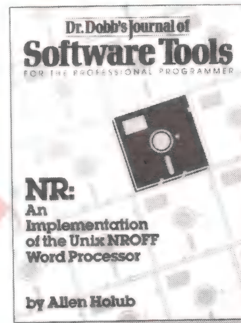
On Command:
Writing a Unix-Like Shell for MS-DOS
by Allen Holub

This book and ready-to-use program demonstrate how to write a Unix-like shell for MS-DOS, with techniques applicable to most other programming languages as well. The book and disk include a detailed description and working version of the Shell, complete C source code, a thorough discussion of low-level MS-DOS interfacing, and significant examples of C programming at the system level.

Supported features include: read, aliases, history, redirection and pipes, Unix-like command syntax, MS-DOS-compatible prompt support, C-Shell-based shell scripts, and a Shell variable that expands to the contents of a file so a program can produce text that is used by Shell scripts. The Unix-like control flow includes: if/then/else; while; foreach; switch/case; break; continue.

The ready-to-use program and all C source code are included on disk. For IBM PC and direct compatibles.

Book & Disk (MS-DOS)
Item #29-1 **$39.95**

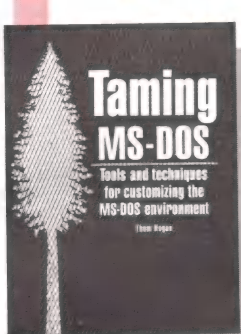


NR:
An Implementation of the Unix NROFF Word Processor
by Allen Holub

NR is a text formatter that is written in C and compatible with UNIX's NROFF. Complete source code is included in the **NR** package so that it can be easily customized to fit your needs. NR also includes an implementation of the -ms macro package and an in-depth description of how -ms works. **NR** does hyphenation and simple proportional spacing. It supports automatic table of contents and index generation, automatic footnotes and endnotes, italics, boldface, and more. **NR** also contains: extensive macro and string capability; number registers in various formats; diversions and diversion traps; input and output line traps.

NR is easily configurable for most printers. Both the ready-to-use program and full source code are included. For PC compatibles.

Manual & Disk (MS-DOS)
Item#33-X **$29.95**

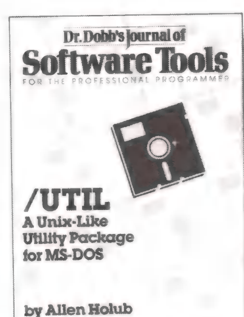


Taming MS-DOS
by Thom Hogan

Taming MS-DOS takes you beyond the basics, picking up where your MS-DOS manual leaves off. You'll learn how to maximize your batch files with routines using redirection, filters and pipes, and routines that prevent accidental reformatting of your hard disk, redefine function keys, and locate files within subdirectories. You'll also learn how to create configurable AUTOEXEC.BAT files, and how to customize CONFIG.SYS and use ANSI.SYS to change the appearance of MS-DOS. **Taming MS-DOS** includes ready-to-use assembly language programs that enhance MS-DOS.

The programs, including batch files and MS-DOS enhancements, are available on disk with full source code.

Book & Disk (MS-DOS)
Item #59-3 **$34.95**
Book **Item #24-0** **$19.95**



Util
by Allen Holub

When used with the Shell, this collection of utility programs and subroutines provides you with a fully functional subset of the Unix environment. Many of the utilities may also be used independently. You'll find executable versions of cat; cp; date; du; echo; grep; ls; mkdir; mv; p; pause; printevn; rm; rmdir; sub; and chmod.

Util includes complete source code on disk and all programs (and most of the utility subroutines) are fully documented in a Unix-style manual. For IBM PC and direct compatibles.

Manual & Disk (MS-DOS)
Item #12-7 **$29.95**

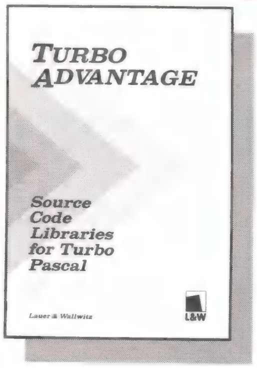
OUR GIFT TO YOU!

Receive a FREE MUG When You Order This Special Holiday Package!

Save 15%
Receive **On Command, Util** and **NR** together for only $85.95!
You get all the convenience of Unix-like features and save 15%.

Unix-like Features Package
Item #167 **$85.95**

Turbo Pascal Tools



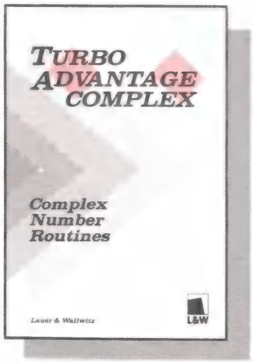
TURBO Advantage:
Source Code Libraries for Turbo Pascal

by Lauer & Wallwitz

The **TURBO Advantage** library contains more than 220 routines complete with source code, sample programs and documentation. Routines are organized and documented under the following categories: bit manipulation; file management; MS-DOS support; sorting routines and spline integration; string operations; and more!

A detailed manual includes a description of each routine, an explanation of the methods used, the calling sequence, and a simple example. Source code is included. For MS-DOS systems.

Manual & Disk (MS-DOS)
Item #26-7 **$49.95**



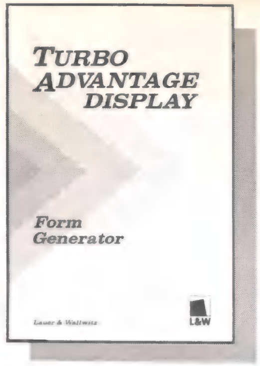
TURBO Advantage Complex:
Complex Number Routines for Turbo Pascal

by Lauer & Wallwitz

TURBO Advantage Complex provides procedures for performing all the arithmetic operations and necessary real functions with complex numbers. Each procedure is based on predefined constants and types. By using these declarations, the size of arrays are easily adapted. Each type declaration is a record with both a real and an imaginary part. Use these procedures to build more sophisticated functions in your own programs. **TURBO Advantage Complex** also demonstrates the use of these procedures in routines for vector and matrix calculation with complex numbers and variables; simultaneous Fourier transforms; calculations of convolution and correlation functions; and more.

Source code and documentation included. For MS-DOS systems. Some of the **TURBO Advantage Complex** routines are most effectively used with routines contained in **TURBO Advantage**.

Manual & Disk (MS-DOS)
Item #27-5 **$89.95**

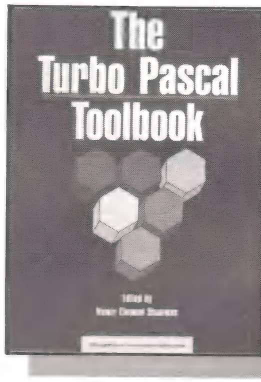


TURBO Advantage Display:
Form Generator for Turbo Pascal

by Lauer & Wallwitz

The **TURBO Advantage Display** form generator includes an easy-to-use form processor, 30 Turbo Pascal procedures and fuctions to facilitate linking created forms to your program, full source code, and documentation. For MS-DOS systems. Some of the **TURBO Advantage** routines are necessary to compile **TURBO Advantage Display**.

Manual & Disk (MS-DOS)
Item #28-3 **$69.95**



The Turbo Pascal Toolbook
Edited by
Namir Clement Shammas

Featuring over 800K of fully documented Turbo Pascal source code, **The Turbo Pascal Toolbook** includes: an extensive library of low-level routines; external sorting and searching tools, presenting a new database routine that combines the best features of the B-, B+, and B++ trees; window management tips; and much more. All routines, libraries, and sample programs are on disk for MS-DOS systems.

Book & Disk (MS-DOS)
Item #61-5 **$45.95**
Book only
Item #25-9 **$25.95**

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT 871 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

M&T Books
501 Galveston Dr.
Redwood City, CA 94063

OUR GIFT TO YOU!

Receive a FREE MUG When Your Order This Special Holiday Package!

Receive TURBO Advantage together with TURBO Advantage Complex or TURBO Advantage Display and SAVE 20%

TURBO Advantage & TURBO Advantage Complex
Item #070A **$115**
TURBO Advantage & TURBO Advantage Display
Item #070B **$99.95**

```
          call  update
          mov    cx,65535
;
;       square root procedure via 8086, integer in CX
;
boundit:  mov     ax,1                    ;infimum - will be lower bound
          mov     dx,cx                   ;supremum - will be upper bound
;
; the next section gets the max lower bound and the min upper bound
          REPT    8
          shl     ax,1                    ;mpy infimum by 2
          cmp     ax,dx                   ;above supremum ?
          ja      root                    ; then done
          shr     dx,1                    ;if not, div supremum by 2
          mov     bx,dx                   ;store supr.
          mov     si,ax                   ;store infim.
          ENDM
;
root:     add     bx,si                   ;get avg. of bounds for
          shr     bx,1                    ;first guess of root
          mov     ax,cx
;
;       Newton's Method =>  X = (Xo + N/Xo)/2
;
Newton:
          xor     dx,dx           ;prepare for division
          div     bx              ;ax still has target
          add     bx,ax           ;newton
          shr     bx,1
;
; we now have a square root in bx
;
;to print, remove the next 8 semicolons
;         mov    ax,cx
;         call   dec_out
;         mov    dl,' '
;         call   char_out
;         mov    ax,bx
;         call   dec_out
;         crlf
didit:   loop boundit
;
done:     call update
          crlf
          xor     al,al
          mov     ah,4Ch
          int     21h
;
;
;         output a hex word in decimal
;
;         CX,AX,DX destroyed
;
dec_out   proc    near
          xor     cx,cx
another:  inc     cx
          xor     dx,dx
          div     base                    ;base is 10 decimal!
          push    dx                      ;remainder is less sig digits
          or      ax,ax                   ;is the quotient zero?
          jnz     another                 ;if not, more number to convert
print_dig:
          pop     dx                      ;retrive digit from stack
          add     dl,'0'                  ;ascii offset
          call    char_out
          loop    print_dig               ;do all of the digits
          ret
dec_out   endp
;
;         output a single character
;
char_out  proc    near
          mov     ah,2                    ;output char function
          int     21h                     ;do it
          ret
char_out  endp
;
;
update    proc    near
          mov     ah,2ch                  ;get dos time
          int     21h                     ;hour in ch, mins in cl,secs in dh
          mov     uper,cl
          mov     secs,dh
          mov     hnds,dl
          mov     al,ch
          xor     ah,ah
          call    dec_out
;
          time_print uper,':'
;
          time_print secs,':'
;
          time_print hnds,'.'
;
          crlf
          ret
update    endp
;
code      ends
          end     sqrt
```

End Listings

Listing One *(Text begins on page 54.)*

```
Listing 1

/* POPWIN.I: Routines for pop-up windows                               */
/* Externally defined are gotoxy(), wrtcha(), window(), chattr(),      */
/*         and videomode().                                            */
/* Notes: Programmer is responsible for assuring that the pop-up       */
/*          defined in the POPDESCR structure is large enough to       */
/*          contain the text.                                          */
/*         Save your cursor position before calling these routines.    */
/*          They don't preserve or restore caller's cursor.            */
/*         Unlike menubar, the POPDESCR structure doesn't contain a    */
/*          pointer to the text elements. Thus, you can use the        */
/*          routines to create dialog boxes and help windows.          */
/* ------------------------------------------------------------------- */
typedef struct {              /* window descriptor/control structure */
  int   left, top, right, bottom;    /* window location, inclusive */
  char  textAttr;                      /* text attribute in window */
  int   border;                 /* 0 = none, 1 = single, 2 = double */
} POPDESCR;

static bord[][6] = {                            /* border characters */
            { 196, 179, 218, 191, 217, 192 },
            { 205, 186, 201, 187, 188, 200 }
};

void popMake (POPDESCR *win)                 /* create pop-up window */
/* Must initialize the POPDESCR structure before calling.              */
/* Does not write text to the window, but only creates it. After       */
/*   this fcn returns, you can write text using popPuts() and          */
/*   control the cursor with popxy(), both given below.                */
{
int  x, y, style, a;

  a = win->textAttr;
  window (win->left, win->top, win->right, win->bottom,
         win->textAttr);                               /* open window */
  if ((style = win->border-1) >= 0)          { /* draw border outside */
    gotoxy (win->left-1, win->top-1, 0);
    wrtcha (bord [style][2], a, 0);            /* corners: upper left */
    gotoxy (win->right+1, win->top-1, 0);
    wrtcha (bord [style][3], a, 0);                    /* upper right */
    gotoxy (win->right+1, win->bottom+1, 0);
    wrtcha (bord [style][4], a, 0);                    /* lower right */
    gotoxy (win->left-1, win->bottom+1, 0);
    wrtcha (bord [style][5], a, 0);                     /* lower left */
    for (x = win->left; x <= win->right; x++) {        /* horizontals */
      gotoxy (x, win->top-1, 0);
      wrtcha (bord [style][0], a, 0);
      gotoxy (x, win->bottom+1, 0);
      wrtcha (bord [style][0], a, 0);
    }
    for (y = win->top; y <= win->bottom; y++) {          /* verticals */
      gotoxy (win->left-1, y, 0);
      wrtcha (bord [style][1], a, 0);
      gotoxy (win->right+1, y, 0);
      wrtcha (bord [style][1], a, 0);
    }
  }
  gotoxy (win->left, win->top, 0);
} /* ------------------------ */
void popScroll (POPDESCR *win)              /* scroll pop up one line */
{
  winScroll (win->left, win->top, win->right, win->bottom,
            win->textAttr);
  gotoxy (wherex (0), wherey (0) - 1, 0);
} /* ------------------------ */
void popxy (int x, int y, POPDESCR *w)  /* gotoxy for popup window */
/* Allows you to express text coordinates relative to upper left       */
/* corner of the window in video page 0                                */
{
int  row, col;

  row = w->top + y;
  col = w->left + x;
  gotoxy (col, row, 0);
} /* ------------------------ */
void popPuts (int x, int y, char string[],       /* write string to */
             POPDESCR *win)                     /* specified window */
{
int   len, ch, p;

  popxy (x, y, win);                 /* position cursor in window */
  len = strlen (string);
  for (ch = 0; ch < len; ch++) {
    if (string [ch] == '\n') {                      /* handle newline */
      x = 0;
      ++y;
      popxy (x, y, win);
      if ((y + win->top) > win->bottom) {     /* scroll if required */
        popScroll (win);
        --y;
      }
    } else {
      if ((x + win->left) > win->right) {       /* outside window */
        x = 0;                                   /* so wrap cursor */
        ++y;
        popxy (x, y, win);
        if ((y + win->top) > win->bottom){ /* scroll if required */
          popScroll (win);
          --y;
        }
      }
      wrtcha (string [ch], win->textAttr, 0); /* write next char */
      ++x;
      popxy (x, y, win);                        /* advance cursor */
    }
  }
} /* ------------------------ */
char *saveScrn (void)                         /* saves screen image */
/* Call this routine before popMake(). It saves the current screen     */
/*   image at a location pointed to by the returned value. You must    */
/*   pass the same pointer back to restScrn() later in order to        */
/*   make the pop-up go away.                                          */
{
int      c;
unsigned srcSeg;
char     *saveArea;

  saveArea = (char *) malloc (4096);             /* allocate space */
  if (videomode (&c) == 7)
    srcSeg = 0xB000;                           /* &monochrome buffer */
  else
    srcSeg = 0xB800;                        /* &text graphics buffer */
  movedata (srcSeg, 0, _SS, (unsigned) saveArea, 4096);    /* save */
  return (saveArea);                              /* return pointer */
} /* ------------------------ */
void restScrn (char *saveArea)            /* restores screen image */
/* Call this routine when you want the pop up window to go away.       */
/* It restores the screen to its appearance before the window. It      */
/*   does NOT restore the cursor. That is your responsibility.         */
/* This routine does not worry about snow on IBM's poorly designed     */
/*   CGA board. Snow may result when restoring the screen.             */
{
int      c;
unsigned destSeg;

  if (videomode (&c) == 7)
    destSeg = 0xB000;                          /* &monochrome buffer */
  else
    destSeg = 0xB800;                        /* text graphics buffer */
  movedata (_SS,(unsigned) saveArea, destSeg, 0, 4096); /* restore */
  free (saveArea);                             /* de-allocate space */
} /* ------------------------ */
```

End Listing One

Listing Two

```
Listing 2

/* MENUBAR.I: Constructs a menu bar per MENUBARSPEC structure          */
/* Externally defined are activepage(), videomode(), wrtstra()         */
/* Notes: Preserve your cursor position before calling this fcn.       */
/*          It does not save or restore caller's cursor.               */
/*        The 'sel' pointer points to a solid string of menu           */
/*          selections in the form "sel1\0sel2\0sel3\0...seln\0"       */
/* ------------------------------------------------------------------- */
typedef struct {
  int       background, foreground;      /* colors used in menu bar */
  int       nsels;                         /* number of selections */
  char      *sel;        /* pointer to static selections (see above) */
} MENUBARSPEC;                     /* caller sets up as many as needed */
                     /* does not remember previous cursor position */

void menubar (MENUBARSPEC *spec)
{
int   p, i, ncols, start, interval, page;
char  attr;

  page = activepage ();                            /* get active page */
  videomode (&ncols);                           /* get screen width */
  gotoxy (0, 0, page);                                  /* go home */
  attr = chattr(spec->foreground, spec->background); /* attributes */
  for (p = 0; p < (ncols-1); p++) {
    wrtcha (' ', attr, page);                       /* blank menu bar */
    gotoxy (wherex (page)+1, 0, page);                   /* advance */
  }
  interval = ncols / spec->nsels;        /* spacing between entries */
  start = i = 0;
  for (p = 0; p < spec->nsels; p++) {            /* write selections */
    gotoxy (start, 0, page);                        /* place cursor */
    wrtstra (&(spec->sel[i]), attr, page);
    i += strlen (&(spec->sel[i])) + 1;           /* find next string */
    start += interval;                     /* next position for entry */
  }
}
```

End Listing Two

Listing Three

```
Listing 3

/* MENUDEMO.C: Demonstrates principles of menu bars and pop-down       */
/*             menus in Turbo C                                        */
/* ------------------------------------------------------------------- */
/* INCLUDES */
#include <dos.h>
#include <video.i>
#include <colors.h>
#include <menubar.i>
#include <popwin.i>

/* LOCAL FUNCTION PROTOTYPES */
void popFileMenu (POPDESCR *mfile, char *text);
void popEditMenu (POPDESCR *medit, char *text);
```

(continued on page 108)

Instant Replay

Version III

See us at
COMDEX/Fall '87
November 2–6, 1987
Bally's Las Vegas B210
Las Vegas, Nevada

SOFTWARE AHEAD OF ITS TIME

"Instant Replay™ is one of those products with the potential to go from unknown to indispensable in your software library." *PC Magazine*

"Incredible . . . We built our entire Comdex Presentation with Instant Replay.™" *Panasonic*

"Instant Replay™ brings new flexibility to prototypes, tutorials, & their eventual implementation." *Electronic Design*

"I highly recommend Instant Replay.™" *Computer Language*

"You need Instant Replay™!" *Washington Post*

Instant Replay™ **is a unique authoring system for creating:**

Demonstrations · Tutorials · Music Presentations · Prototypes · Menus

When active, *Instant Replay*™ is a DOS Shell that runs, memorizes, and replays programs. It has the unusual ability to insert prompts, pop-up windows, prototypes, user involvement, music, and branching menus into replays.

Building a Demo or Tutorial is easy. Just run any program with *Instant Replay*™ and insert explanation pop-up windows, prompts, user involvement, music and so on. *Instant Replay*™ remembers everything; it builds a demo that will re-run the actual program or a screens only prototype version.

Instant Replay™ includes a *Screen Maker* for building pop-up windows, prototype windows, and menu windows. Other useful tools include a *Prototyper, Keystroke Editor, Music Maker, Menu Maker, Presentation Text Editor, Control Guide, and Insertion Guide.*

The screen editor *Screen Genie*™ is designed to be memorized by *Instant Replay.*™ Creating animations is easy and fun; just run *Screen Genie*™ and memorize your activity.

Instant Replay™ for IBM and True Compatibles, requires DOS 2.0 or greater. *Instant Replay*™ is not copy protected. **There are no royalties required for distribution of Demos.**

Instant Replay™ at $149.95 is an exciting new product. Because of the quality of this product, *Nostradamus*® provides a 60-day satisfaction money back guarantee. Call or write, we accept VISA, AmEx, C.O.D., Check or P.O. with orders. Demo diskettes and free product brochure available.

Nostradamus, Inc. 3191 S. Valley Street, (ste. 252) Salt Lake City, Utah 84109
voice (801) 487-9662
Data/BBS 801-487-9715 1200/2400,n,8,1

Instant Replay™ Features:

- *Illustrated manual with index*
- *Music Maker*
- *Text Editor*
- *Online Help*
- *Screen Painter*
- *Magic Demo Animator*
- *Dynamic Menu Maker*
- *Memorize and Replay actual programs*
- *Memorize and Replay screens only*
- *Insert: prompts, pop-ups, prototypes, music, and user involvement into replays*
- *Make insertions while creating or reviewing*
- *Generate Vapor Ware from actual programs*
- *Resident Screen Painter for creating and grabbing windows on the fly*
- *Prototyper that includes slide shows, menus, and nesting*
- *Keystroke/time editor, inserter, and merger*
- *Replay chaining and linking*
- *Modular demo making facilities*
- *Fast forward and single step modes*
- *Self Made Tutorial included*
- *Timed Keyboard Macros*
- *Numerous and powerful operator input options for Tutorials*
- *Text File Presentation facility*
- *Transparent Windows*
- *Change Defaults*
- *Foreground or Background Music*
- *Canned special sound effects*
- *Unlimited replay branching*
- *Compressed screens*
- *Object oriented programming*
- *Tracking editor*
- *Plus much more . . .*

Instant Replay™
Instant Assistant™
Screen Genie™
Word Genie™
NoBlink Accelerator™
Assembler Genie™
DOS Assistant™
Programming Libraries
Supports Turbo Pascal 4.0, Turbo C, MS Quick C
HardRunner™
. . . and more

Nostradamus®

Listing Three

(Listing continued, text begins on page 54.)

```
/* STATICS FOR PROGRAM */
char entries[] = {"File\0Edit\0Browse\0Reports\0Exit\0"};
MENUBARSPEC menuspec = {{BLUE}, {WHITE}, {5}, {&entries}};
char fileMenu[] = {"Open\nSave\nNew file\nAbandon"};
char editMenu[] = {"Select record\nRemove field\nAdd field"};
/* --------------------------*/
main ()
{
static POPDESCR filePop = {{1}, {2}, {8}, {5}, {0}, {2}};
POPDESCR   editPop;
char       *prevScrn;
int        n, oldx, oldy, cols;

/* SET UP POP-DOWN MENU FOR FILE SELECTION */
  filePop.textAttr = chattr (YELLOW, BLUE);

/* SET UP POP-DOWN MENU FOR EDIT SELECTION */
  editPop = filePop;
  editPop.left = 80 / menuspec.nsels + 1;
  editPop.right = editPop.left + 12;
  editPop.bottom = editPop.top + 2;

/* NOW SHOW THE TWO MENUS IN SUCCESSION */
  setmode (3);
  cls ();
  menubar (&menuspec);                              /* write menu */
  printf ("\nActive page is %d", activepage()); /* show video info */
  printf ("\nVideo mode is %d", videomode (&cols));
  printf ("\nNumber of columns on screen is %d", cols);
  oldx = wherex (activepage());
  oldy = wherey (activepage());
  printf ("\nCursor is at x = %d, y = %d", oldx, oldy);
  gotoxy (oldx, oldy, activepage());
  getch ();                                    /* wait for keypress */
  cursoff ();
  popFileMenu (&filePop, fileMenu);           /* pop down file menu */
  popEditMenu (&editPop, editMenu);
  setcursor (0, cursend());                 /* make a block cursor */
  gotoxy (oldx, oldy, activepage());
  puts ("\nPress any key to end demo...");
  getch ();
  setcursor (cursend()-1, cursend());  /* restore underline cursor */
  cls ();
} /* ------------------------ */
void popFileMenu (POPDESCR *mfile, char *text)
{
char   *prevScrn;
  prevScrn = saveScrn ();
  popMake (mfile);
  popPuts (0, 0, text, mfile);
  getch ();
  restScrn (prevScrn);
} /* ------------------------ */
void popEditMenu (POPDESCR *medit, char *text)
{
char   *prevScrn;

  prevScrn = saveScrn ();
  popMake (medit);
  popPuts (0, 0, text, medit);
  getch ();
  restScrn (prevScrn);
} /* ------------------------ */
```

End Listings

New Horizons...for dBASE Users.

Resident...Compatible...Powerful – FrontRunner offers all this and more.

- CREATE MEMORY RESIDENT dBASE PROGRAMS. FrontRunner is the first memory-resident applications development tool which uses the popular dBASE programming language.
- BUILT-IN COMPILER. FrontRunner applications can be compiled into binary files for protected distribution and unparalleled speed.
- dBASE III+ LANGUAGE AND FILE COMPATIBILITY. Standard commands, functions, and syntax are provided, along with complete data and index file compatibility, allowing you to use FrontRunner immediately.
- UNIQUE POWERKEY FEATURE. Bind commands or entire programs to a single PowerKey for rapid execution from within other applications.
- PASTE COMMAND. This powerful command allows extraction of data into your application, using standard filtering expressions.

dBASE and dBASE III+ are registered trademarks of Ashton-Tate, Inc.

The MS-DOS version of FrontRunner is $295.00. Orders received before December 1, 1987 include an unlimited run-time license, allowing you to distribute royalty-free applications throughout the universe. (After this date the unlimited run-time package will be sold separately.) FrontRunner is not copy-protected, and comes with a 30-day money back guarantee. To order, call the number below and get a running start toward more exciting and productive applications development!

APEX

Apex Software Corporation
4516 Henry Street
Pittsburgh, PA 15213
(412) 681-4343

Now Available From Ashton-Tate, Inc.

dBASE Power...On Demand.

CIRCLE 185 ON READER SERVICE CARD

Listing One *(Text begins on page 126.)*

```
                    Listing 1 -- kernel.h, Printed 9/11/1987

#ifndef NULL
#include <stdio.h>
#endif
#include <tools/pq.h>

                                /* Error codes                                  */
#define TE_NOERR          0  /* No error                                     */
#define TE_TOOMANY       -1  /* Maximum number of tasks (32) already exists */
#define TE_NOMEM         -2  /* Insufficient memory available               */
#define TE_BADARG        -3  /* Illegal Argument                            */
#define TE_TIMEOUT       -4  /* Timeout                                     */
#define TE_QFULL         -5  /* Queue is full                               */
#define TE_NOTASKS       -6  /* No tasks to send message                    */
#define TE_INTERNAL      -7  /* Internal error                              */
#define TE_DEADLOCK      -8  /* Delete would have caused a deadlock.        */
#define TE_STACK         -9  /* Stack overflow                              */
#define TE_KILL         -10  /* Ctrl-Break encountered                      */

#define TS_NORMAL         0  /* Must be 0                                   */
#define TS_WAIT           1
#define TS_TIMEOUT        2

#define T_MAXTASK        32 /* Max. number of tasks that can be active    */
#define TQ_SIG       0xa5a5 /* Signature used for queues to test validity */

/* PRIORITY(a,b) evaluates to a neagive number if task a is lower priority
 *               than task b, to 0 if they're equal, to a positive number
 *               if task a is higher priority than task b. If priorities
 *               are the same, the timestamps are compared and the routine
 *               with the smaller (older) time stamp is assumed to be the
 *               higher priority.
 */

#define T_PRIORITY(a,b)  ( ((a)->priority != (b)->priority)          \
                            ? (a)->priority  - (b)->priority          \
                            : (b)->timestamp - (a)->timestamp         )

/*----------------------------------------------------------------------
 *  Task Control Block. Do not change the register-save area
 *  (ax, bx, cx ... ) without also changing the code in swap.asm.
 *  Don't change anything without changing the offset to the stack
 *  base in chkstk.asm.
 *
 *  I'm assuming the small model here. That is, I'm assuming that
 *  the only segment register that can change is the extra segment
 *  and that the stack and data segments always have the same value.
 *
 *  Be sure to block() if you're going to modifiy the CS, DS, or SS
 *  registers.
 *
 *  A context swap is done by pushing the registers in the following
 *  order:
 *              flags,cs,ip,ax,bx,cx,dx,si,di,bp,ds,es
 *
 *  Then, the current stack pointer is saved in the TCB. Context is
 *  restored by popping es,ds,bp,di,si,dx,cx,bx, and ax, and then
 *  restoring the flags, cs, and ip with an iret instruction.
 */

typedef struct tcb
{
    void        **sp;           /* Must be first and must be 16 bits   */
    unsigned    ss;             /* Must be second &  must be 16 bits   */

    unsigned    priority;       /* priority 0=lowest, 65,535=highest    */
    unsigned long timestamp;    /* Clock tick when task was preempted.  */

    unsigned    wait;           /* Counting semaphore used by tasks that
                                 * are waiting at a queue. Set to initial
                                 * timeout value and decremented on each
                                 * clock tick.  Task is put back into
                                 * the active list if semaphore gets to
                                 * 0. If wait < 0, task will not time out.
                                 */

    struct tcb  *next;          /* Pointer to next task waiting at queue. */

    int         status;         /* TS_NORMAL  Not suspended by wait.
                                 * TS_WAIT    Suspended by wait
                                 */


    void        *msg;           /* Dequeued message if task was waiting
                                 * for a message. NULL if task timed out.
                                 */

                                /* The following are handy for debugging */
                                /* but aren't used for anything else     */
    char        *tag;           /* Identifying string of some sort       */
    void        **initial_sp;   /* Initial stack pointer                 */

    void        *stack[1];      /* First cell of stack. Must be last
                                 * thing in the structure. Must be declared
                                 * as pointer-sized for t_create().
                                 */
}
TCB;

typedef struct t_queue
{
    int         signature;      /* Signature                            */
    struct t_queue *next;       /* Next queue in chain.                 */
    TCB         *task_h;        /* Head (start) of task list.           */
    TCB         *task_t;        /* Tail (end)   of task list.           */
    int         q_size;         /* Maximum number of elements           */
    int         numele;         /* # of elements currently in queue     */
    void        **headp;        /* Head pointer                         */
```

(continued on page 113)

Oasys presents

Microsoft® *cross* C

for VAX, Sun, Apollo . . .

VAX
SUN
APOLLO

OASYS is proud to announce the immediate availability of the OASYS/Microsoft Cross C Development System. Microsoft C, MASM (Assembler), and LINK (Linker) now run on DEC VAX (VMS and Ultrix), Sun and Apollo systems.

Those accustomed to using these superior Microsoft tools on a PC can now build MS-DOS applications on a VAX or workstation. OASYS guarantees that the unsurpassed speed, compactness, and flexibility of Microsoft C have been preserved. The OASYS/Microsoft Cross C Development System offers identical functionality to Microsoft C -- no short-cuts, no alterations -- repackaged to meet today's demands for high performance/low cost development on non-MS-DOS systems.

With the OASYS/Microsoft Cross C Development System you can maintain, or even extend, applications originally created on a PC. Software development teams can now build large, complex MS-DOS (soon OS/2) applications on powerful centralized VAXs or networked workstations.

Regardless of where you choose to do development, OASYS provides the best tools, on the widest variety of hosts, with comprehensive support. Our exclusive relationship with Microsoft, the world's leading supplier of MS-DOS software products, is evidence of our commitment to provide evolving PC tools to OASYS customers.

Prices start at $1,000. New ports are underway. Call today for more information. OEM and end-user inquiries are encouraged.

Oasys

Microsoft®

230 Second Avenue, Waltham, MA 02154 **(617) 890-7889**

Listing One *(Listing continued, text begins on page 126.)*

```
    void            **tailp;      /* Tail pointer                     */
    void            *queue[1];    /* First cell of actual queue.
                                   * Must be at the bottom of the
                                   * structure.
                                   */
}
T_QUEUE;

/*----------------------------------------------------------------------
 *      Global variables. Actually declared in globals.c. I'm assuming
 *      the default initialization to 0 here. These may be used by
 *      your programs (T_clock and T_numtasks are useful) but should
 *      never be modified by them. It's safest to block while
 *      accessing them.
 */

#ifdef ALLOC
#       define CLASS
#       define I(x) x
#else
#       define CLASS extern
#       define I(x)
#endif

CLASS PQ   *T_tasks I(=0);   /* Priority queue of tasks that are waiting     */
                             /* for service. See /src/tools/pq.c for priority */
                             /* queue routines and definition of PQ.          */

CLASS TCB  *T_active I(=0);  /* Pointer to currently active task. NULL if
                              * multitasking is off or if no tasks are active
                              * (this latter is a deadlock).
                              */

CLASS unsigned long T_clock I(=0);

                             /* Incremented on each system clock tick. If you
                              * assume the default 18.2 ticks/second,
                              * the clock will roll over after about
                              * 65552 hours (about 7.47 years):
                              *
                              *    ((0xffffffff/18.2) /60) /60 == 65552
                              *    65552 / 24 /365.35          == 7.47798
                              *
                              * Of course, this number will scale with faster
                              * tick rates but the resolution should be ok for
                              * all reasonable tick rates.
                              */

CLASS T_QUEUE *T_queues   I(=0); /* Pointer to head of linked list of queues. */
CLASS int      T_numtasks I(=0); /* Total # of tasks that have been created.  */

/*----------------------------------------------------------------------
 * Function prototypes. You should never call any of the _t_xxxx
 * functions directly.
 */

extern  T_QUEUE *t_makequeue   (int size                               );
extern  int     t_send         (T_QUEUE *q,    void *msg               );
extern  void    *t_wait        (T_QUEUE *q,    int timeout             );
extern  int     t_yield        (void                                   );
extern  int     t_perror       (char *str,     int errcode             );
extern  int     t_start        (int factor                             );
extern  TCB     *t_create      (int (*subr)(), char* tag, unsigned pri,
                                               int stk_size,...);
extern  int     t_chg_priority (TCB *tp,       int new_priority        );
extern  int     t_delete       (TCB *task                              );
extern  int     t_print        (TCB *task                              );
extern  void    t_stop         (int exit_code                          );
extern  int     t_second       (void                                   );
extern  void    _t_swap        (TCB *old, TCB *new                     );
extern  void    _t_install     (TCB *new                               );
extern  void    _t_shazam      (void                                   );
```

End Listing One

Listing Two

```
                Listing 2 -- schedule.asm, Printed 9/11/1987

    PAGE    56,132
    TITLE   SPEEDUP.ASM: System-clock-modification routines
;-----------------------------------------------------------------

DEBUG    equ 1    ; Set to 1 to make internal symbols public for
                  ; debugging.

DOSPEED  equ 1    ; Set to 0 to disable everything except
                  ; global-variable initialization in speedup.

;-----------------------------------------------------------------
; Public Subroutines are:
;
; t_cli()
;       Disable Interrupts
; t_sti()
;       Enable Interrupts
;
;-----------------------------------------------------------------
;  t_speedup( factor )
; int factor;
;
```

(continued on next page)

Listing Two

(Listing continued, text begins on page 126.)

```
;       Speed up the system clock by the indicated factor
;       (1, 2, 3, 4, etc.).  Call the scheduler
;       on every timer interrupt and call the default clock
;       routine as well every "factor" ticks. For example,
;       speedup( 2 ) speeds up the system clock by
;       a factor of two; the normal interrupt-service
;       routine that's used by DOS will be called every-
;       other tick. A speedup factor of 1 or 0 doesn't modify
;       the clock rate.
;
; _t_slowdown()
;
;       Restore the clock to the normal speed and disconnect
;       the routine installed with a previous speedup() call
;       (if one is installed).
;-------------------------------------------------------------
;
; t_block()
; t_release()
;
;   Disable the scheduler but not the normal clock interrupt.
;   The default system interrupt-service routine is processed
;   in the normal way, on every Nth clock tick. Use these
;   routines carefully. If they're used in a tight loop, it's
;   possible that ALL timer interrupts will be ignored, even
;   if the processor is released for a while inside the loop.
;   In this situation sti() and cli() are better.
;
;-------------------------------------------------------------
; long numint();            Number of unblocked interrupts.
; long numblk();            Number of blocked interrupts.
;

_TEXT   SEGMENT  BYTE PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT  WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP    CONST,  _BSS,   _DATA
        ASSUME   CS: _TEXT, DS: DGROUP, SS: DGROUP, ES: DGROUP

EXTRN   __chkstk:NEAR
EXTRN   __t_reschedule:NEAR
EXTRN   _T_active:WORD

;-------------------------------------------------------------

TIMR_CTRL    = 43H        ; address of timer control port
TIMR_0_DATA  = 40H        ; address of counter 0 data port
TIMR_0_LOAD  = 36H        ; control word for timer

STKSIZE = 256             ; Number of bytes on interrupt-
                          ; service-routine stack

;-------------------------------------------------------------

_TEXT     SEGMENT

;-------------------------------------------------------------
; Misc. variables. Note that I'm putting all these in the
; code (_TEXT) segment so that I can find them when an
; interrupt comes along. The PUBLIC statements are just for
; debugging.

old_int     equ  $
old_off     dw   ?        ; Offset of old timer interrupt routine.
old_seg     dw   ?        ; Segment address of same.

tick_reset  dw   ?
numticks    dw   ?        ; Initialized to tick_reset, decre-
                          ; mented on each timer interrupt,
                          ; reset to the speedup factor (and the
                          ; old service routine is called) when
                          ; it reaches zero.

stack  db  STKSIZE dup (0) ; Local stack for service routine
                           ; 30 bytes are used by real service
                           ; routine, the rest is available
                           ; for the user service routine.
stack_end   dw  ?

old_ds      dw  ?
old_sp      dw  ?
old_ss      dw  ?
old_ax      dw  ?
old_ip      dw  ?
old_cs      dw  ?
old_fl      dw  ?

numint      dd 0          ; total number of interrupts
numblk      dd 0          ; number of times user routine blocked

blocked     db  0         ; don't execute user routine if true

IF DEBUG
    PUBLIC  old_int, old_off, old_seg, old_ds,
    PUBLIC  tick_reset, numticks, stack, stack_end, old_sp,
    PUBLIC  old_ss, old_ax, old_ip, old_cs, old_fl
    PUBLIC  blocked, serv, numint, numblk
ENDIF

;-------------------------------------------------------------
; statistics stuff.
;
PUBLIC _t_numint, _t_numblk

_t_numint PROC NEAR
        mov     ax,WORD PTR cs:numint
        mov     dx,WORD PTR cs:numint+2
        ret
_t_numint ENDP

_t_numblk PROC NEAR
        mov     ax,WORD PTR cs:numblk
        mov     dx,WORD PTR cs:numblk+2
        ret
_t_numblk ENDP

;-------------------------------------------------------------
; t_cli();  t_sti();  Disable and enable interrupts.
;
PUBLIC  _t_cli, _t_sti

_t_cli  PROC NEAR
        cli
        ret
_t_cli  ENDP

_t_sti  PROC NEAR
        sti
        ret
_t_sti  ENDP


;-------------------------------------------------------------
; t_block();     Disable and enable user interrupt service
; t_release();   routine but not the real interrupt service
;                routine (or the interrupt itself).

PUBLIC  _t_block, _t_release

_t_block PROC NEAR
        mov   byte ptr cs:blocked,1
        ret
_t_block ENDP

_t_release PROC NEAR
        mov   byte ptr cs:blocked,0
        ret
_t_release ENDP

;-------------------------------------------------------------
;  t_speedup( factor )
; int factor;
;                                        factor  = [bp+4]
;                                        routine = [bp+6]
PUBLIC  __t_speedup

__t_speedup     PROC NEAR

        push    bp
        mov     bp, sp
        xor     ax,ax
        call    __chkstk

        mov     _TEXT:old_ds,ds     ; remember current DS.
        mov     ax,[bp+4]           ; AX          = factor
        mov     _TEXT:tick_reset,ax ; tick_reset  = factor;
        mov     _TEXT:numticks,ax   ; numticks    = factor;

if DOSPEED

        mov     ax,[bp+4]           ; if( factor && factor != 1)
        cmp     ax,01H              ; {
        je      noload              ;
        cmp     ax,00H              ;
        je      noload              ;

        mov     al,TIMR_0_LOAD      ;     Set up timer for load
        out     TIMR_CTRL,al        ;
        mov     ax,00000H           ;     Number of ticks
        mov     dx,00001H           ;           = 65536/factor
        mov     bx,[bp+4]           ;     BX = factor.
        div     bx                  ;     AX = number of ticks
        cli                         ;
        out     TIMR_0_DATA,al      ;     Send new count to timer
        mov     al,ah               ;
        out     TIMR_0_DATA,al      ;
        sti                         ; }
noload:
                                    ; Get the old vector
        mov     ah,35H              ;
        mov     al,08H              ;
        int     21H                 ;
        mov     _TEXT:old_off,bx    ;
        mov     _TEXT:old_seg,es    ;

                                    ; set up the new vector
        mov     ah,25H              ;
        mov     al,08H              ;
        mov     dx,OFFSET _TEXT: serv ;
        push    ds                  ;
        push    cs                  ;
        pop     ds                  ;
        int     21H                 ;
        pop     ds                  ;
endif
        mov     sp,bp
        pop     bp
        ret

__t_speedup     ENDP

;-------------------------------------------------------------

PUBLIC  __t_slowdown

__t_slowdown    PROC NEAR

        push    bp
        mov     bp, sp
        xor     ax,ax
        call    __chkstk
```

```
        mov     ax,_TEXT:old_off ; See if the interrupts have
        or      ax,ax            ;                   changed.
        jz      no_int           ; No, don't fix them then

                                 ; restore old timer interrupt
        push    ds               ;
        mov     ah,25H           ;
        mov     al,08H           ;
        mov     ds,_TEXT:old_seg ;
        mov     dx,_TEXT:old_off ;
        int     21H              ;
        pop     ds               ;

no_int:
        mov  al,TIMR_0_LOAD      ; Restore default system
        out  TIMR_CTRL,al        ; clock tick rate
        mov  al,0
        out  TIMR_0_DATA,al
        out  TIMR_0_DATA,al

        mov     sp,bp
        pop     bp
        ret

__t_slowdown    ENDP

;-------------------------------------------------------------
; Actual interrupt service routine.
; Note that the flags, cs, and ip are pushed on entry (because
; if the interrupt)
;
serv    PROC    NEAR

        push    ax

        mov     al, byte ptr cs:blocked ; If( servicing blocked)
        or      al,al                   ; {
        jz      serv1                   ;
                                        ;
        add     WORD PTR cs:numblk,1    ;    ++numblk;
        adc     WORD PTR cs:numblk+2,0  ;
        jmp     servexit                ; }
serv1:                                  ; else
        add     WORD PTR cs:numint,1    ; {
        adc     WORD PTR cs:numint+2,0  ;    +numint;
        push    bx                      ;
        push    cx                      ;    Save rest of
        push    dx                      ;        current context
        push    si
        push    di
        push    bp
        push    ds
        push    es

        mov     ds,_TEXT:old_ds         ;    Restore Data segment
        mov     bx,_T_active            ;    BX = T_active
        mov     [bx],sp                 ;    T_active->sp = SP
        mov     [bx+2],ss               ;    T_active->ss = SS

        push    cs                      ;    Set up local stack
        pop     ss                      ;
        mov     sp,offset _TEXT: stack_end ;

        call    __t_reschedule          ;   in task.c

        mov     bx,_T_active            ;   BX = new T_active,
        mov     ss,[bx+2]               ;   will be the same as
        mov     sp,[bx]                 ;   the old T_active if
        pop     es                      ;   no change is reqd.
        pop     ds                      ;
        pop     bp                      ;
        pop     di                      ;
        pop     si                      ;
        pop     dx                      ;
        pop     cx                      ;
        pop     bx                      ;
                                        ;
servexit:                               ; }
        dec     _TEXT:numticks          ; if(--numticks > 0)
        jle     serv3                   ; {
        mov     al,20h                  ;    send EOI
        out     20h,al                  ;
        pop     ax
        iret                            ; }
                                        ; else
serv3:                                  ; {
        mov     ax,_TEXT:tick_reset     ;    numticks = tick_reset;
        mov     _TEXT:numticks,ax       ;
        pop     ax                      ;
        jmp     dword ptr _TEXT:old_int ;    jmp to old vector
                                        ; }
serv    ENDP


_TEXT   ENDS
END
```

End Listing Two

(Listing Three begins on page 116.)

Listing Three

(Listing continued, text begins on page 126.)

```
                    Listing 3 -- swap.asm, Printed 9/11/1987

; SWAP.ASM      Routine to do context swaps. Everything in
;               this file is VERY compiler dependant and
;               VERY nonportable.
;
        TITLE   swap.c
        NAME    swap

_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT  WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'

op      db       0       ; Used by rst_chkstk and chg_chkstk
segm    dw       0       ; (below).
off     dw       0

save_bx dw       0       ; used by __t_swap_in

_BSS    ENDS
DGROUP  GROUP   CONST, _BSS, _DATA
        ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP

_DATA      SEGMENT
_DATA      ENDS

IF 1
PUBLIC stack_err, mychkstk, rst_chkstk, chg_chkstk
PUBLIC op, segm, off
ENDIF

;--------------------------------------------------------------
_TEXT      SEGMENT

        ASSUME  CS: _TEXT

PUBLIC   __t_swap_in     ; Swaps two tasks
PUBLIC   __t_install     ; Installs a task when none active
PUBLIC   __t_shazam      ; Starts multitasking
PUBLIC   _t_stop         ; Stops  multitasking.
PUBLIC   __t_sus_chkstk  ; Temporarily suspend stack checking.
PUBLIC   __t_rst_chkstk  ; Restore it again.

EXTRN    __chkstk:NEAR   ; in standard library
EXTRN    _free:NEAR      ; In standard library
EXTRN    __t_slowdown:NEAR; In schedule.asm
EXTRN    _t_block:NEAR   ; "
EXTRN    _t_release:NEAR ; "
EXTRN    _t_iserr:NEAR   ; In task.c
EXTRN    _T_active:WORD  ; Declared in kernel.h. Pointer
                         ; to currently active task

;--------------------------------------------------------------

save_sp dw 0
save_ss dw 0
chk_on  dw 0             ; No stack probes while nonzero

;--------------------------------------------------------------

__t_shazam PROC NEAR

    ; Start the ball rolling. Save the current context.
    ; then start up the first task. On entry,
    ; T_active must point at the first task to activate.
    ; Stack on entry (from top to bottom) is:
    ;
    ;  return address from _t_shazam call
    ;  old bp saved by t_start
    ;  t_start's return address
    ;  speedup_factor passed to t_start
    ;

    add sp,2                ; Discard return addr to _t_shazam
                            ; Uncovering bp from main that
                            ; was saved by t_start()
    push si
    push di
    push ds                 ; push this last

    mov  WORD PTR cs:save_sp, sp
    mov  WORD PTR cs:save_ss, ss

    call chg_chkstk
    mov  bx, WORD PTR _T_active
    jmp  shazam

__t_shazam ENDP

;--------------------------------------------------------------

_t_stop PROC NEAR

    ; t_stop( errcode )
    ;
    ; This routine deletes the current task, causes
    ; multitasking to be turned off, and passes control
    ; back to the routine that called t_start()
    ; (immediately following the t_start call).
    ; Errcode is passed back to the calling routine as the
    ; return value of t_start().
    ;
    ; It can be called directly by a running task; it's
    ; called automatically when the last task is deleted,
    ; or when the only running task deletes itself.

    cli                                 ; Just to make sure

    add sp,2                        ; Discard return address
    pop ax                          ; return value = errcode

    mov ss, WORD PTR cs:save_ss ; Restore initial stack...
    mov sp, WORD PTR cs:save_sp ;
    pop ds                          ; ...and data segment.

    push  ax
    call  rst_chkstk                ; Put back original __chkstk
    call  __t_slowdown              ; Disable weird timer int.

    pop   ax                        ; get back return value
    push  ax

    or    ax,ax                     ; if( errcode == NOERR ) )
    jnz   t_stopl                   ; {
    push  _T_active                 ;       free( T_active );
    call  _free                     ;
    add   sp,2                      ; }

t_stopl:
    pop   ax                        ; return( errcode )

    pop   di                        ; resore si and di saved by
    pop   si                        ; by __t_shazem,
    pop   bp                        ; and bp saved by t_start.
    ret                             ;
_t_stop ENDP

;--------------------------------------------------------------

__t_install PROC NEAR

    ; _t_install(new)
    ; TCB       *new;
    ;
    ;  Delete the current task and replace it with the
    ;  new one. This routine saves some space (and execution
    ;  time) by jumping into the middle of swap() to install
    ;  the new task. The scheduler must be blocked when
    ;  this routine is called. This routine does not
    ;  return.

    add   sp,2                      ; discard return address
    pop   bx                        ; bx = new

    mov   ss,WORD PTR [bx+2]        ; ss:sp = new task's stack;
    mov   sp,WORD PTR [bx]

    push  bx
    push  _T_active                 ; free( T_active )
    call  _free
    add   sp,2                      ; Discard arg to free()
    pop   bx;                       ; get back bx.

    jmp   shazam

__t_install ENDP

;--------------------------------------------------------------

__t_swap_in PROC NEAR

    ; _t_swap_in( new )
    ; TCB *new;
    ;
    ; Do a context swap. Replace T_active with new, This
    ; routine returns only when the original context is
    ; restored. Swapping MUST be blocked when this subroutine
    ; is called. Release() is called once the new context
    ; is installed and T_active is modified to point at the
    ; new task.

    cli
    mov   WORD PTR cs:save_bx,bx
    pop   bx              ; bx = return address
    pushf                 ; Save current context
    push  cs
    push  bx              ;     (Push return address as new ip)
    push  ax
    push  save_bx
    push  cx
    push  dx
    push  si
    push  di
    push  bp
    push  ds
    push  es

    ; Stack now looks like this:
    ;
    ;   new      [sp + 24]
    ;   flags    [sp + 22]
    ;   cs       [sp + 20]
    ;   ip       [sp + 18]
    ;   ax       [sp + 16]
    ;   bx       [sp + 14]
    ;   cx       [sp + 12]
    ;   dx       [sp + 10]
    ;   si       [sp + 8]
    ;   di       [sp + 6]
    ;   bp       [sp + 4]
    ;   ds       [sp + 2]
    ;   es       [sp]             (top of stack)

    mov   bx,WORD PTR _T_active
    mov   WORD PTR [bx+2],ss
    mov   WORD PTR [bx],sp

    mov   bx,sp
    mov   bx,WORD PTR [bx+24]   ; bx = new

shazam:                             ; __t_shazam and __t_install
                                    ; come here to do the swap
```

```
    mov   WORD PTR _T_active,bx ; T_active = new;
    mov   ss,WORD PTR [bx+2]     ; Switch to new task's stack
    mov   sp,WORD PTR [bx]

    pop es
    pop ds
    pop bp
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax

    call _t_release
    sti
    iret

__t_swap_in ENDP


;-----------------------------------------------------------------
; __t_chg_chkstk()                Normal stack checking off
; __t_rst_chkstk()                back on again
;
; __t_sus_chkstk()                Suspend stack checking temporarily
; __t_rst_chkstk()                restore it again.
;
; Turn off Microsoft stack checking by overwriting the first
; 5 bytes of __chkstk with an absolute jump to mychkstk.
; This is a kludge but I can't get the Microsoft compiler
; to link my own version of __chkstk, even when I use the
; source file that they supply.

chg_chkstk PROC NEAR

    mov bx,OFFSET __chkstk

    mov ah,BYTE PTR cs:[bx+0]
    mov BYTE PTR op,ah

    mov ax,WORD PTR cs:[bx+1]
    mov WORD PTR off,ax

    mov ax,WORD PTR cs:[bx+3]
    mov WORD PTR segm,ax
    mov BYTE PTR cs:[bx+0],0EAH                 ; EA=JMP
    mov WORD PTR cs:[bx+1],OFFSET mychkstk      ; offset
    mov WORD PTR cs:[bx+3],cs                   ; segment

    mov cs:chk_on,1                       ; Enable stack checking
    ret

chg_chkstk ENDP

rst_chkstk PROC NEAR

    mov bx,OFFSET __chkstk

    mov ah,BYTE PTR op
    mov BYTE PTR cs:[bx+0],ah

    mov ax,WORD PTR off
    mov WORD PTR cs:[bx+1],ax

    mov ax,WORD PTR segm
    mov WORD PTR cs:[bx+3],ax
    ret

rst_chkstk ENDP

__t_sus_chkstk PROC NEAR
    mov cs:chk_on,0
    ret
__t_sus_chkstk ENDP

__t_rst_chkstk PROC NEAR
    mov cs:chk_on,1
    ret
__t_rst_chkstk ENDP

;-----------------------------------------------------------------
; On entry AX holds the number of bytes required for local
; variables. Chkstk normally checks the stack and, at the
; same time finishes setting up the stack frame by
; subtracting the contents of ax from the stack pointer.
;

mychkstk PROC   NEAR

    mov cx,cs:chk_on     ; If stack checking disabled at
    or  cx,cx            ; run time, skip past the actual
    jz  nocheck          ; test.

    mov cx,_T_active
    add cx,44            ; Offset to stack base + 4
    cmp sp,cx            ; if( sp <= stack_base )
    jbe stack_err

nocheck:
    pop cx               ; cx = return address
    sub sp,ax            ; finish setting up stack frame
    jmp cx               ; ret to caller w/o modifying stack.

stack_err:               ; Return address still on the stack
    mov  ax,-9
    push ax
    call _t_stop         ; Shouldn't return

```

(continued on next page)

Listing Three

(Listing continued, text begins on page 126.)

```
        mov cx,0
        jmp cx                  ; Panic abort to DOS

mychkstk ENDP

_TEXT   ENDS
END
```

End Listing Three

Listing Four

Listing 4 -- globals.c, Printed 9/11/1987

```
#define ALLOC 1

#include "kernel.h"
```

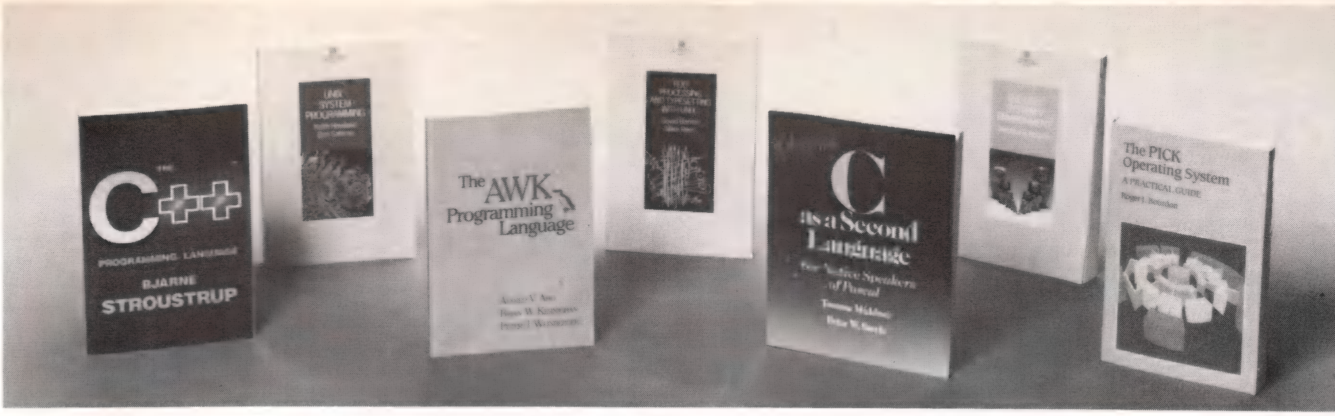
End Listing Four

Listing Five

Listing 5 -- queue.c, Printed 9/11/1987

```
#include "kernel.h"

/*----------------------------------------------------------------*/

T_QUEUE *t_makequeue( size )
int      size;
{
     /* Make a queue for intertask communication and link it
      * into the the queue chain. Return values are  pointer
      * to queue on suceess or TE_NOMEM if Insufficient memory.
      * Queues are searched in order of creation so it's best
      * to create the most active queues first. Be sure that
      * multitasking is blocked when tailp is modified (because
      * it's static).
      */

     static T_QUEUE *tailp;
     T_QUEUE        *p;
     void           *malloc();

     t_block();
     p = (T_QUEUE *) malloc( sizeof(T_QUEUE)
                              + ((size-1) * sizeof(void *)) );
     t_release();

     if( !p )
          return (T_QUEUE *) TE_NOMEM;

     p->signature  = TQ_SIG;
     p->next       = NULL;
     p->task_h     = NULL;
     p->task_t     = NULL;
     p->numele     = 0;
     p->q_size     = size;
     p->headp      = p->queue;
     p->tailp      = p->queue;

     t_block();

     if( !T_queues )
          T_queues = tailp = p ;
     else
     {
          tailp->next = p;
          tailp       = p;
     }

     t_release();
     return p;
}

/*----------------------------------------------------------------*/

int      t_send( q, msg )
T_QUEUE *q;              /* Pointer to queue                  */
void     *msg;           /* Pointer to message to enqueue     */
{
     /* Send a message and reschedule if necessary.
      *
      * Return Values:
      *    TE_NOERR        No error;
      *    TE_BADARG       Bad q argument.
      *    TE_QFULL        Queue is full
      *
      * The message is always enqueued in the indicated queue.
      * Then, if a task is waiting, The message at the head of
      * the queue is dequeued and attached to the task, which
      * is put back into the active list. Finally, if a task
      * was activated, the current process yields. Note,
      * however, that the current task will still be the
      * active task if it's higher priority than the one to
      * which you send a message. The sending task should wait()
      * somewhere to make room for the lower-priority task.
      */

     TCB *task;

     if( q->signature != TQ_SIG )
          return( TE_BADARG );

     t_block();

     if( q->numele == q->q_size )          /* Queue is full */
     {
          t_release();
          return( TE_QFULL );
     }

     /*
      * Enqueue the message.
      */

     ++ q->numele;
     if( ++q->tailp >= q->queue + q->q_size )
          q->tailp = q->queue ;
     *(q->tailp) = msg ;


     if( q->task_h )
     {
          /* A task is waiting, dequeue both it and the message,
           * attach the message to the task, and reschedule
           */

          task      = q->task_h;
          q->task_h = task->next;

          --q->numele;
          if( ++q->headp >= q->queue + q->q_size )
               q->headp = q->queue;

          task->msg    = *(q->headp);
          task->status = TS_WAIT    ;
          pq_ins( T_tasks, &task )  ;

          t_yield();
     }

     t_release();
     return TE_NOERR;
}

/*----------------------------------------------------------------*/

void     *t_wait( q, timeout )
T_QUEUE  *q;
int      timeout;
{
     /* Wait for a message to arrive at the queue. If several
      * tasks are waiting at the same queue, the first task
      * in the queue gets the message. Return if no message
      * arrives within timeout system clock ticks.  Maximum
      * timeout is 32,767 ticks. A 0 timeout value
      * means that the subroutine returns immediately
      * (without a reschedule) if no message is waiting
      * in the queue.
      *
      * Message requests are queued up in order recieved,
      * without reguard to priority. I've done this both
      * because it's easy and because, in most applications,
      * tasks with different priorities will not be pending
      * on the same queue.
      *
      * If a message is present, the routine returns it
      * imediately without yielding, otherwise the current
      * task is removed from the active list and yield()
      * is called.
      *
      * Hints: Use this routine to suspend a task for a
      * limited amount of time (as compared to deleteing
      * the task). Just pend on a queue that will never have
      * a message sent to it.
      *
      * Normally, a pointer to the message is returned, other
      * return values are:
      *
      *    TE_TIMEOUT  on a timeout or if the input value of
      *                timeout is 0 and no message is waiting
      *
      *    TE_NOTASKS  There current task is the only one in
      *                existance. This is a guaranteed deadlock.
      */

     TCB  *new;

     if( q->signature != TQ_SIG )
          return( TE_BADARG );

     t_block();

     if( q->numele )
     {
          /* There's a message waiting in the queue. Dequeue
           * the message and return it immediately. Strictly
           * speaking, we don't have to attach the message
           * to the task, but it's convenient to do it for
           * debugging reasons.
           */

          -- q->numele;
          if( ++q->headp >= q->queue + q->q_size )
               q->headp = q->queue ;

          T_active->msg    = *(q->headp);
          T_active->status = TS_WAIT ;

          t_release();
```

(continued on page 120)



All systems are go!

When you get your operating systems books from the industry leader.

☐ **C As a Second Language for Native Speakers of Pascal**—Tomasz Müldner and Peter Steele, *both of Acadia University*
Here's a complete and detailed description of the C programming language and programming techniques in C.
(19210) 456 pp. Paper 1987 $26.95

☐ **The C Book**—Michael Banahan, *The Instruction Set, Ltd.*
This tutorial, based on Banahan's training courses, features a structured approach to C.
(17370) 304 pp. Paper 1988 $24.75*
***Order this book today, and we'll ship it to you as soon as it's available (January '88)!**

☐ **The C++ Programming Language**—Bjarne Stroustrup, *AT&T Bell Laboratories*
The definitive reference to the C++ language, an extension of C that Stroustrup developed at AT&T Bell Laboratories.
(12078) 328 pp. Paper 1986 $29.25

☐ **The PICK Operating System: A Practical Guide**—Roger J. Bourdon, *Aston Technology Ltd.*
Here's a solid introduction to the management of a PICK-based computer system, covering all the major features of PICK.
(18055) 450 pp. Paper 1987 $32.25

☐ **The UNIX® System V Environment**—Stephen R. Bourne, *Digital Equipment Corporation*
Written by a key member of the group that developed the UNIX system, here's a gold mine of information on UNIX toolkits, along with a definitive reference to the new standard, System V.
(18484) 378 pp. Paper 1986 $26.95

☐ **Text Processing and Typesetting with UNIX**—David Barron, *University of Southampton*, and Michael Rees, *University of Tasmania*
Written in a clear tutorial style, this book offers a complete description of the capabilities of UNIX text processing and typesetting for a wide audience.
(14219) 250 pp. Paper 1987 $26.95

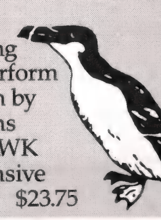
☐ **UNIX System Programming**—Keith Haviland and Ben Salama, *both of Sphinx Ltd.*
Here's an advanced programming book for software developers producing applications written in C for the UNIX system.
(12919) 480 pp. Paper 1987 $26.95

☐ **Operating Systems: A Systematic View, Third Edition**—William S. Davis, *Miami University, Ohio,*
An update of the popular book that gives you a straightforward introduction to operating systems.
(11185) 539 pp. Hardcover 1987 $37.75

☐ **An Introduction to Operating Systems, Revised First Edition**—Harvey M. Deitel, *Boston College*
In addition to complete coverage of the fundamental concepts of operating systems, this best-seller includes five case studies—on UNIX, VAX/VMS, CP/M, MVS, and VM.
(14501) 673 pp. Hardcover 1984 $44.25

☐ **Operating Systems Concepts, Second Edition**—James L. Peterson and Abraham Silberschatz, *both of the University of Texas at Austin*
This popular book clearly and concisely defines the fundamental concepts of operating systems and their application to *any* system.
(06198) 625 pp. Hardcover 1985 $44.25

OF SPECIAL INTEREST…
☐ **The AWK Programming Language**—Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger, *all of AT&T Bell Laboratories*
Here's the authoritative reference on the newest version of AWK, a pattern-matching language for writing short programs to perform common data-manipulation tasks. Written by AWK's original developers, the book begins with a tutorial that shows you how easy AWK is to use, and continues with a comprehensive manual. (07981) 210 pp. Paper 1987 $23.75



UNIX is a registered trademark of AT&T.

To order, check off the books that interest you, tear out this ad and send check for total, plus your local sales tax, to: Dept. DM, Addison-Wesley Publishing Co., Reading, MA, 01867-9984. Or simply call the number below.

Addison-Wesley Publishing Company
Reading, Massachusetts 01867 • (617) 944-3700

16024

CIRCLE 195 ON READER SERVICE CARD

Listing Five

(Listing continued, text begins on page 126.)

```

        return T_active->msg;
    }
    else                                /* No messages waiting  */
    {
        if( timeout == 0 )              /* Immediate time out   */
        {
            t_release();
            return TE_TIMEOUT;
        }
        else
        {
            /* Enqueue the current task to wait for an
             * incomming message. The pq_del call
             * gets a task to preempt the current one.
             */

            if( !pq_del( T_tasks, &new) )
            {
                t_release();            /* No tasks to activate */
                return TE_NOTASKS;
            }
            else
            {
                T_active->wait      = timeout;
                T_active->next      = NULL;
                T_active->timestamp = T_clock;

                if( !q->task_h )
                    q->task_h = T_active;
                else
                    q->task_t->next = T_active;

                q->task_t = T_active;

                _t_swap_in( new );  /* Returns on either a message
                                                          being */
                                    /* sent to this queue or a
                                                     timeout.  */

                return ( T_active->msg ? T_active->msg : TE_TIMEOUT
                                                                 );
            }
        }
    }
}

/*--------------------------------------------------------------*/

t_yield()
{
    /* Yield to the highest priority task (if there is one).
     * You can't yield to tasks waiting at queues, only to
     * active ones.
     *
     * Returns:
     *  TE_NOERR     successfull yield
     *  TE_NOTASKS   Current task is the only active task
     */

    TCB *new, *old ;

    t_block();

    if( ! pq_del( T_tasks, &new ) )
    {
        t_release();
        return TE_NOTASKS;
    }

    old = T_active;

    old->timestamp = T_clock ;
    pq_ins( T_tasks, &old );

    _t_swap_in( new );  /* _t_swap_in() changes T_active to new; */
    return TE_NOERR ;
}
```

End Listing Five

Listing Six

```
                Listing 6 -- task.c, Printed 9/11/1987

#include <dos.h>
#include <stdarg.h>
#include <signal.h>
#include <tools/hardware.h>     /* #define for TIMR_CLK */
#include "kernel.h"

static int   Speedup_factor = 0;
static long  Executed, Timed_out, Did_swap;

#define max(a,b)  ((a) > (b) ? (a) : (b))

/*-----------------------------------------------------------
 * Strings for error codes. Note that these are upside down
 * to compensate for the negative indexes
 */

static char     *msgs[] =
{
    "Ctrl-Break or Ctrl-C"                                 /* -10 */
    "Stack overflow",                                      /*  -9 */
    "Delete would have caused a deadlock",                 /*  -8 */
    "Internal error",                                      /*  -7 */
    "No tasks to send message to",                         /*  -6 */
    "Queue is full",                                       /*  -5 */
    "Timeout",                                             /*  -4 */
    "Illegal Argument",                                    /*  -3 */
    "Insufficient memory available",                       /*  -2 */
    "Maximum number of tasks (32) already exists",         /*  -1 */
    "No error"                                             /*   0 */
};

char **t_errlist = msgs + ((sizeof(msgs)/sizeof(*msgs)) -1);

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

t_iserr(x)
{
    return( -10 <= (x) && (x) < 0 );
}

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

t_perror( str, errcode )
char    *str;
{
    if( !t_iserr(errcode) && errcode != 0 )
        t_printf( "%s status %d\n", str, errcode );
    else
        t_printf( "%s %s\n", str, t_errlist[errcode] );
}

/*--------------------------------------------------------------*/

static intr()
{
    t_stop( TE_KILL );
}

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

int     t_start( speedup_factor )
{
    /* Start multitasking. At least one task must have
     * been created prior to this call. The speedup factor
     * determines the system clock-tick rate. A value of 1
     * gives the default rate of roughly 18.2 times a second
     * (once every 55 milliseconds, more or less). A
     * speedup_factor of 2 gives twice that speed: 36.4 ticks
     * per second, one every 27 milliseconds or thereabouts.
     * Speeding up the clock rate shouldn't affect the DOS
     * clock. Nonetheless, it's safest if the speedup factor
     * is a power of two.
     *
     * If the speedup factor is 0 then the system is
     * nonpreemptive. You'll have to use t_yield(), t_send(),
     * and t_wait() to change contexts.
     *
     * Control passes imediately to the highest priority task.
     * Control will automatically pass back to the calling
     * subroutine when all tasks have been deleted. The task
     * is actually started in _t_shazam(), declared in swap.asm.
     *
     * Normal return values:
     *
     *  TE_NOTASKS      No tasks exist, multitasking not
     *                  started;
     *
     *  TE_DEADLOCK     A task deleted itself and it's the
     *                  only active task in the system. Other
     *                  tasks exist but they're all pending
     *                  on queues.
     *
     *  TE_NOERR        All tasks have been deleted normally,
     *                  no tasks are waiting on queues.
     *
     *  TE_STACK        Task stack overflow.
     *
     * If TE_NOERR is returned, then all memory allocated to
     * tasks will have been saved, otherwise, if one of the
     * above errors was returned, T_active will point at the
     * TCB of the offending task.
     *
     * Other return values are possible if a task calls t_stop()
     * directly. The argument passed to t_stop() is returned by
     * t_start(). The process is analogous to the value of
     * exit(), which doesn't return and who's argument is
     * passed back to a wait() call in the parent process. Note
     * that the TCB pointed to by T_active will not be free()ed
     * unless TE_NOERR (0) is returned.
     */

    Speedup_factor = speedup_factor;

    if( !pq_del( T_tasks, &T_active ) )
        return TE_NOTASKS;

    if( speedup_factor > 0 )
    {
        t_cli();
        _t_speedup( speedup_factor );
    }

    signal( SIGINT, intr );

    _t_shazam();
    return TE_INTERNAL;          /* Shouldn't ever get here */
}

/*--------------------------------------------------------------*/

int     t_second()
{
    /* Returns the number of system clock ticks in a
```

```
     * second, given the speedup factor passed to t_start().
     * Returns 0 if the speedup factor was 0. In this case
     * a t_wait() call will never time out.
     */

    return (TIMR_CLK / 65536) * Speedup_factor ;
}

/*-------------------------------------------------------------*/

TCB     *t_create( subr, tag, priority, stack_size, ... )

int      (*subr)();      /* Subroute that forms main module */
char     *tag;           /* String used to identify TCB     */
unsigned priority;       /* Priority                        */
int      stack_size;     /* Stack size (in 2-byte words)    */
{
    /* Creates a new task. Subr is a pointer to the main()
     * subroutine for the task.
     *
     * Priorities must be in the range 0-255. 255 is the
     * highest. If more than one task has the same priority,
     * they are executed in a round-robin
     * fashion. Forces a reschedule if tasking is active.
     *
     * Arguments may be passed to the subroutine at startup.
     * That is, a NULL-terminated list of pointer-sized
     * arguments follow stack_size in the t_create() call.
     * These are passed to the subroutine in the normal way.
     * For example:
     *
     * foo( a, b, c ) int a, b, c; {}
     * t_create( foo, "foo" 10, 128, doo, wha, ditty, NULL);
     *
     * starts up foo() as a task at priority 10 with a
     * 128-byte stack. Doo, wha, and ditty are passed
     * to foo as arguments a, b, and c. Note that the
     * arguments use 6 of the 128 bytes in the stack
     * (two for each argument). The "foo" tag is just
     * used for identification purposes in debugging.
     * It can be any string.
     *
     * Note that a few Microsoft functions (like printf)
     * use up inordinate amounts of stack. If you're going
     * to call Microsoft library routines, you'll need
     * at least 1K bytes of stack (stack_size==512) per
     * task.
     *
     * A pointer to the created TCB is returned normally.
     * Error return values are:
     *
     * TE_TOOMANY Maximum number of tasks already exists
     * TE_NOMEM   Insufficient memory available
     */

    TCB     *t;
    struct  SREGS segs;
    int     pq_cmp();
    int     pq_swap();
    va_list argptr;
    void    *arg;
    void    *malloc();

    va_start( argptr, stack_size );

    if( ++T_numtasks > T_MAXTASK )
        return (TCB *) TE_TOOMANY;

    t_block();

    /*  Allocate the stack, converting stack_size to bytes.
     *  I'm requesting one more cell than specified in order
     *  to make room for the error return address, below
     *  [stack[0] is included in the sizeof(TCB)]. The minimum
     *  stack size is 20 words, this gives us enough for a
     *  context swap plus a little slop.
     */

    stack_size = max( stack_size, 20 );

    if( !(t = (TCB *) malloc(sizeof(TCB)+(stack_size*sizeof(void*))
                                                                 )))
    {
        t_release();
        return (TCB *) TE_NOMEM;
    }

    /* Create the active queue if necessary, then initialize
     * the TCB. The stack pointer is initialized to point
     * just past the end of the stack (rather than to the
     * last cell) because a push uses a predecrement. The
     * PC points at the subroutine. Uninitiailzed registers
     * are unimportant, but will contain 0. The stack area
     * is initialized to the pattern a5a5a5a5.... so that
     * we can look it with a debugger and see what's been
     * used. 0 is no good for this purpose because 0 is
     * a likely thing to be pushed on the stack.
     */

    if( !T_tasks )
        T_tasks = pq_create( T_MAXTASK, sizeof(TCB *),
                                        pq_cmp, pq_swap, NULL);
    segread( &segs );

    memset( t,        0x0,  sizeof(TCB) - sizeof(void*) );
    memset( t->stack, 0xa5, stack_size  * sizeof(void*) );

    t->sp         = t->stack +  ++stack_size;
    t->ss         = segs.ds  ;  /* Stacks are in data seg */
    t->initial_sp = t->sp    ;
    t->priority   = priority ;
    t->timestamp  = T_clock  ;
    t->tag        = tag      ;

```

(continued on next page)

Listing Six

(Listing continued, text begins on page 126.)

```
    /* Initialize the stack, First pretend that we've
     * already called the initial subroutine by pushing
     * the arguments, and t_stop as a dummy return address.
     * The task shouldn't return, but if it does, t_stop
     * seems like a reasonable thing to call, even though
     * it will return garbage.
     * The last 11 things are the initial context.
     * They'll be popped as part of the context swap.
     */

    while( arg = va_arg(argptr, void*) )
        *--(t->sp) = arg;

    *--(t->sp) = t_stop;        /* Vector to t_stop */
    *--(t->sp) = 0;             /* flags*/
    *--(t->sp) = segs.cs;       /* cs   */
    *--(t->sp) = subr;          /* ip   */
    *--(t->sp) = 0;             /* ax   */
    *--(t->sp) = 0;             /* bx   */
    *--(t->sp) = 0;             /* cx   */
    *--(t->sp) = 0;             /* dx   */
    *--(t->sp) = 0;             /* si   */
    *--(t->sp) = 0;             /* di   */
    *--(t->sp) = 0;             /* bp   */
    *--(t->sp) = segs.ds;       /* ds   */
    *--(t->sp) = segs.es;       /* es   */

    if( T_active  &&  T_PRIORITY( t, T_active ) <= 0 )
    {
        pq_ins( T_tasks, &T_active );
        _t_swap_in( t );
    }
    else
    {
        pq_ins( T_tasks, &t );
        t_release();
    }

    return t;
}

/*------------------------------------------------------------*/

static TCB      *del_fm_queues( task )
TCB             *task;
{
        /* Traverse the queue list and if the task is waiting
         * for a message, delete it from the queue
         * and return a pointer to it, otherwise return NULL.
         */

        T_QUEUE *q;
        TCB     *t, **prev ;

        for( q = T_queues; q ; q = q->next )
        {
            prev = &q->task_h;

            for( t = q->task_h; t; prev = &t->next, t = t->next)
            {
                if( t == task )
                {
                    *prev = t->next ;
                    return t;
                }
            }
        }

        return NULL;
}

/*------------------------------------------------------------*/

int     t_chg_priority( tp, new_priority )
TCB     *tp;
int     new_priority;
{
    /* Change priority for indicated task. Forces a reschedule.
     * If the task was waiting on a message, it is immediately
     * timed out and put back on the active list. A task may
     * change it's own priority.
     *
     * Return values:
     *
     * TE_NOERR
     * TE_BADARG        Task doesn't exist;
     */

    int   rval = TE_NOERR;
    TCB   *deleted;
    int   pq_rm_cmp();

    if( new_priority > 255 )
        return TE_BADARG;

    t_block();

    if( tp == T_active )
    {
        T_active->priority = new_priority;
        t_yield();
    }
    else
    {
        if( !pq_remove( T_tasks, &deleted, pq_rm_cmp, tp ) )
            if( deleted = del_fm_queues( tp ) )
            {
                deleted->status    = TS_TIMEOUT;
                deleted->msg       = NULL;
            }
            else
                return TE_BADARG;

        deleted->priority = new_priority;
        pq_ins( T_tasks, &deleted );
        t_yield();
    }
}

/*------------------------------------------------------------*/

int     t_delete( task )
TCB     *task;
{
    /* Delete task created with previous t_create() call and
     * free all memory associated with task. Note that
     * malloced() memory is not freed, only the memory that
     * t_create() allocated to begin with. A task may delete
     * itself. Forces a reschedule.
     *
     * Return values:
     *
     *  TE_BADARG       Task doesn't exist
     *  TE_NOERR
     *
     * T_stop is called automatically when the only active
     * task deletes itself. See t_start() for an explanation
     * of the return stati.
     *
     * For convenience, a task may delete itself with a
     * t_delete(NULL) call.
     */

    TCB         *deleted;
    static TCB  garbage;
    int         pq_rm_cmp();

    t_block();

    if( T_active == task  || task == NULL )
    {
        /* Delete the current task
         * Note that the t_install() call below will not
         * return. It replaces the current task with the new
         * one, a pointer to which is in "deleted."
         */

        if( !pq_del( T_tasks, &deleted) )
            t_stop( T_numtasks <= 1 ? TE_NOERR : TE_DEADLOCK );
        else
        {
            --T_numtasks;
            _t_install( deleted );
        }
    }
    else
    {
        /* Delete a task that's not active. pq_remove tries
         * to get it from the active list. If that's not
         * successful, del_fm_queues() scans the queues looking
         * for it. If that's not successfule, TE_BADARG is
         * returned.
         */

        if( pq_remove( T_tasks, &deleted, pq_rm_cmp, task ) )
            free( deleted );

        else if( deleted = del_fm_queues(task) )
            free( deleted );
        else
        {
            t_release();
            return TE_BADARG ;
        }

        if( --T_numtasks <= 0 )     /* Deleted the only task */
            t_stop( TE_NOERR );
    }

    t_release();
    return TE_NOERR;
}

/*------------------------------------------------------------*/

static  pq_cmp( task1, task2 )
TCB     **task1, **task2;
{
    return T_PRIORITY( *task1, *task2 );
}

static  pq_swap( task1, task2 )
TCB     **task1, **task2;
{
    TCB *tmp;

    tmp    = *task1;
    *task1 = *task2;
    *task2 = tmp;
}

static pq_rm_cmp( task1, item )
TCB     **task1;
TCB     *item;
{
    return !( *task1 == item );
}

/*------------------------------------------------------------*/

static outc(c)
{
    if( c == '\n' )
        putch('\r');

    putch(c);
```

```
}

t_printf( fmt )
char    *fmt;
{
    /* The doprnt() function used here is from: Allen Holub,
     * The C Companion (Englewood Cliffs: Prentice-Hall,
     * 1987). You can also use the ANSI vprintf(). Note
     * that the Microsoft version of vprintf() uses A LOT
     * of stack. If you're using vprintf, your tasks should
     * have at least 1K bytes of stack.
     */

    va_list     args;

    va_start(args, fmt);

    t_block();
    doprnt(outc, 0, fmt, args); /* vprintf( fmt, args ); */
    t_release();
}

/*------------------------------------------------------------*/

t_sstats()
{
    /* Print various scheduler related statistics. This routine
     * should not be called from a task (because it uses printf).
     */

    printf("\nScheduler called %ld times: %ld Tasks timed_out, "
                                        "%ld context swaps\n",
                              Executed, Timed_out, Did_swap );
}

/*------------------------------------------------------------*/

#pragma check_stack-

TCB *_t_reschedule()
{
    static T_QUEUE      *q;
    static TCB          *t, **prev ;

    /* Workhorse function called by the schedular
     * (timer-interrupt service routine).
     *
     * Scan all the queues, checking for timed-out tasks. If
     * you find one, remove it from the queue and add it to
     * the active list.
     *
     * Modify T_active to point at the next task to activate.
     * (the original T_active if no change).
     */

    ++Executed;

    _t_sus_chkstk();            /* Stack probes off for the nonce */

    for( q = T_queues; q ; q = q->next )
    {
        prev = &(q->task_h);

        for( t = q->task_h ; t ; )
        {
            if( --(t->wait) <= 0 )
            {
                ++Timed_out;
                *prev     = t->next ;
                t->msg    = NULL ;
                t->status = TE_TIMEOUT;
                pq_ins( T_tasks, &t );
            }

            prev = &(t->next);
            t    =  t->next ;
        }
    }

    /* Check the highest-priority element of the queue.
     * If it's not higher than the current task, do nothing.
     * Otherwise do a context swap. pq_replace will
     * extract the highest priority object from the active
     * list and put it into t, simultaneously putting
     * T_active into the list.
     */

    T_active->timestamp = ++T_clock ;

    if( t = *(  (TCB **) pq_look(T_tasks)) )
    {
        if( T_PRIORITY(t, T_active)  >= 0 )
        {
            ++Did_swap;
            pq_replace( T_tasks, &t, &T_active );
            T_active = t ;
        }
    }

    _t_rst_chkstk();            /* Stack probes on again */
}
#pragma check_stack+
```

End Listing Six

Listing Seven

Listing 7 -- tdebug.c, Printed 9/11/1987

```
#include <stdio.h>
#include "kernel.h"

/* TDEBUG.C     Various routines that are useful for debugging but
 *              probably won't end up in the final system.
 */

t_tprint( t )
TCB     *t;
{
    /* Print a TCB */

    char  **p, *str ;
    int   i;

    printf("---------------- <%s> at %04x ---------------\n",
                                                t->tag, t );

    printf("stack %04x:%04x, priority %d, timestamp %d,",
                    t->ss,t->sp, t->priority, t->timestamp );

    printf(" wait %d, status %d\n", t->wait, t->status );

    printf("next %04x, initial sp %04x, msg %04x ",
                            t->next, t->initial_sp, t->msg );
    pstr( t->msg, 10 );

    printf("stack[ 0] @  %04x = %04x,  ",
                            &(t->stack)[0], (t->stack)[0]);
    printf("stack[ 1] @  %04x = %04x\n\n",
                            &(t->stack)[1], (t->stack)[1]);

    /* Print the top 15 elements of the stack */

    i = 15;
    for( p = (char **)t->sp; p < t->initial_sp && --i>=0; p++ )
    {
        printf("   sp[%2d] @  %04x = %04x = ",
                            (char **)t->sp - p, p, *p);
        pstr( *p, 32 );
    }

    if( i < 0 )
        printf("(Stack dump truncated at 15 elements)\n");

    printf("--------------------------------------------\n");
}

/*------------------------------------------------------------*/

static pstr( str, len )
char    *str;
{
    /* Print a string with dots instead of nonprinting
     * characteres
     */

    putchar('<');

    for( len = 32 ; --len>=0 && *str ; str++ )
            putchar( ' ' <= *str && *str < 0x7f ? *str : '.' );

    printf(">\n");
}

/*------------------------------------------------------------*/

t_qprint( q )
T_QUEUE *q ;
{
    /* Print out the contents of a queue */

    TCB *t;
    int i;

    if( q->signature != TQ_SIG )
    {
            printf("Queue is invalid (bad signature)\n");
            return;
    }

    printf("------------- Queue at %04x ---------------\n", q);
    printf("%d/%d messages in queue, next queue at %04x\n",
                            q->numele, q->q_size, q->next );

    printf("Waiting tasks: ");
    if( t = q->task_h )
        printf("(none)\n" );
    else
    {
        for( ; t ; t = t->next )
        {
            printf( "%s, ", t->tag );
            if( t == q->task_t )
                    printf("(end)\n");
        }
        printf("\n");
    }

    printf( "head = &queue[%d], tail = &queue[%d], queue is:\n",
                    q->headp - q->queue, q->tailp - q->queue);

    for( i = 0; i < q->q_size ; i++ )
    {
            printf("queue[%d]: %04x ", i, q->queue[i] );
            pstr( q->queue[i], 32 );
    }
    printf("--------------------------------------------\n");
}
```

End Listings

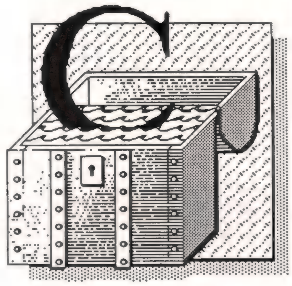
Listing One *(Text begins on page 144.)*

```
( LOAD screen for DDJ Standard Prelude and String Extension)
( MJT  Aug 30 1987 for DDJ December 1987)

(      2 LOAD ( Standard prelude)
       3 LOAD ( Augmented interpretation)
    4  5 THRU ( Controlled words)
    6 13 THRU ( Strings)


( FORTH-83 functions-- typical definitions)
( Adjust these words for your Forth.    See DDJ Oct 1987.)
( Note: functions already provided need not be redefined.)
: RECURSE    [COMPILE] MYSELF ;  IMMEDIATE
: INTERPRET   INTERPRET ;

: I> ( - 'data)   COMPILE R> ;  IMMEDIATE
: >I ( - 'data)   COMPILE >R ;  IMMEDIATE

( Used for alignment: )
: ALIGN     ( HERE 1 AND ALLOT) ;
: REALIGN   ( a - a' )   ( DUP 1 AND +) ;

2 CONSTANT CELL    : CELL+   2+ ;    : CELLS   2* ;

: UNDO    I> R> R> 2DROP >I ;  \ Undoes a DO-- LOOP.
( Required definitions - used to support further compilation)

: THRU ( n n2)   1+ SWAP DO  I LOAD  LOOP ;
\ LOADS screens n through n2.

: \   >IN @ 64 + -64 AND >IN ! ;  IMMEDIATE
\ comment to end of line.  For use in screens only.

: \\   1024 >IN ! ;  IMMEDIATE
\ stops interpreting or compiling screen immediately.

: \IF ( f )   0= IF  [COMPILE] \  THEN ;  IMMEDIATE
\ conditional interpretation or compilation.

: NEED ( - f)   32 ( ie blank) WORD FIND  SWAP DROP  0= ;
\ true if the following word is in the search order.
\ FORTH-83 Controlled Words

NEED  2* \IF  :  2*    DUP  + ;
NEED D2* \IF  : D2*   2DUP D+ ;

NEED HEX \IF  : HEX   16 BASE ! ;
NEED  C, \IF  : C, ( n )   HERE 1 ALLOT C! ;

NEED BL \IF  32 CONSTANT BL

NEED ERASE \IF  : ERASE ( a n)   00 FILL ;
NEED BLANK \IF  : BLANK ( a n)   BL FILL ;

NEED .R \IF  : .R ( n width)   >R DUP 0< R> D.R ;


\ DDJ Forth Column Controlled Words
NEED 2>R
\IF : 2>R   COMPILE SWAP COMPILE >R COMPILE >R ;  IMMEDIATE
NEED 2R>
\IF : 2R>   COMPILE R> COMPILE R> COMPILE SWAP ;  IMMEDIATE
NEED @EXECUTE \IF  : @EXECUTE   @ EXECUTE ;

NEED AGAIN
\IF  : AGAIN   0 [COMPILE] LITERAL [COMPILE] UNTIL ;  IMMEDIATE
NEED DLITERAL
DUP \IF  : DLITERAL  SWAP [COMPILE] LITERAL [COMPILE] LITERAL ;
    \IF   IMMEDIATE


NEED S>D  \IF  : S>D  ( n - d)    DUP 0< ;
NEED WITHIN   \IF  : WITHIN ( n n2 n3 - f)  OVER - >R - R> U< ;
NEED TRUE \IF  -1 CONSTANT TRUE
\ String primitives

: /STRING ( a n n2 - a+n2 n-n2)   ROT OVER +  ROT ROT - ;
\ truncates leftmost n chars of string.  n may be negative.

VARIABLE CTEMP

: CTO"" ( c - a 1)   CTEMP C!  CTEMP 1 ;
\ converts character to string.


\ SKIP and SCAN

: SKIP ( a l c - a2 l2)
\ returns shorter string from first position unequal to byte.
   >R  BEGIN  DUP
       WHILE  OVER C@ R@ - IF  R> DROP  EXIT  THEN  1 /STRING
       REPEAT   R> DROP ;
```

```
: SCAN ( a 1 byte - a2 12)
\ returns shorter string from first position equal to byte.
   >R  BEGIN  DUP
       WHILE  OVER C@ R@ =  IF  R> DROP  EXIT  THEN  1 /STRING
       REPEAT   R> DROP ;


\ String compilation

: PLACE ( a n a2)   2DUP !  1+ SWAP CMOVE ;
\ moves string ( a n ) to be a packed string at a2.

: ASCII ( - c)   \ value of following character.
   BL WORD 1+ C@  STATE @   \ STATE-smart ASCII
   IF  [COMPILE] LITERAL  THEN ;  IMMEDIATE

: ,"   \ compiles following string as packed string at HERE
   ASCII " WORD COUNT  DUP >R  HERE PLACE  R> 1+ ALLOT  ALIGN ;



\ String literals

: (")   I> COUNT  2DUP + >I ;

: " ( - a n)   STATE @   \ string literal.
   IF  COMPILE (") ,"
   ELSE  ASCII " WORD COUNT >R  PAD I CMOVE  PAD R>  THEN ;
  IMMEDIATE


\ Number conversion operator
VARIABLE DPL  \ punctuation locator.

: VAL? ( a n - d 2 , n2 1 , 0)
\ string to number conversion primitive.  True if d is valid.
\ Returns d if number contains ",-./:"  and sets DPL = 0
\ Returns n if no punctuation present   and sets DPL = 0<
   PAD OVER -  SWAP OVER >R  CMOVE
   BL PAD C!  PAD DPL !  0 0 R>  DUP C@ ASCII - = DUP >R - 1-
   BEGIN  CONVERT  DUP C@  DUP ASCII : =
     SWAP ASCII , ASCII / 1+ WITHIN  OR
   WHILE  DUP DPL !  REPEAT  R> SWAP >R IF  DNEGATE  THEN
   PAD 1- DPL @ - DPL !  R> PAD = ( valid?)
   IF  DPL @ 0< IF DROP 1 ELSE 2 THEN  ELSE  2DROP 0  THEN ;


\ -TEXT and COMPARE

: -TEXT ( a n a2 - -1 , 0 , 1)
\ returns -1 if string a n < a2 n , 0 if equal, and 1 if >.
   OVER 0= IF  ROT 2DROP  EXIT  THEN
   SWAP 0 DO  OVER C@ OVER C@ - ( these chars <> ?)
            IF  UNDO  C@ SWAP C@ > 2* 1+  EXIT  THEN  1 1 D+
         LOOP  2DROP 0 ;

: COMPARE  ( a n a2 n2 - -1 , 0 , 1)
\ returns -1 if a n < a2 n2 , 0 if equal, and 1 if >.
   ROT  2DUP ( lengths ) 2>R  MIN SWAP  -TEXT  DUP
   IF    2R> 2DROP
   ELSE  DROP  2R> 2DUP = ( lengths = ?)
       IF  2DROP 0  ELSE  > 2* 1+  THEN
   THEN ;
\ IN

: -MATCH ( a n a2 n2 - ???? -1 , offset 0)
\ returns the position of string a2 n2 in (a n).
\ Offset is zero if ( a n ) is found in first char position.
\ Returns true with invalid offset if ( a n ) isn't in a2 n2.
   2SWAP  2 PICK DUP ( len1 ) >R  OVER SWAP -  DUP 0< R> 0= OR
   IF  2DROP 2DROP  TRUE EXIT  THEN
   0 TRUE ( index match? )  ROT 1+ 0
   DO  DROP ( index ) >R
     2OVER 2OVER DROP  -TEXT 0=  ( equal? )
     IF  R> 0 LEAVE  THEN  1 /STRING  R> 1+  TRUE
   LOOP
   2>R  2DROP 2DROP  2R> ;


\ Useful string operators

: VAL ( a n - d f)   VAL?  DUP 3 < AND
\ converts string to double number.  True if number is valid.
   DUP IF  1 = IF  S>D  THEN  TRUE EXIT  THEN  DUP DUP ;

: EVAL ( a n )
\ evaluates ("text interprets") a string.
   DUP >R  TIB SWAP CMOVE  R@ #TIB !
   0 >IN ! 0 BLK !  INTERPRET  R> >IN ! ;
```

End Listings

A Preemptive Multitasking Kernel and More Mean Subroutines



The main topic of both this and next month's columns is a small preemptive multitasking kernel for the IBM PC (see Listings One–Seven, beginning on page 110). Most of the kernel is written in C, so it shouldn't be too difficult to port it to another environment, including a ROMed environment. The kernel lets you do multitasking within a program—that is, it lets you run subroutines as independent tasks. It also supports a message-passing system that allows for inter-task communication (and time-outs when a message isn't received within a specified time). It doesn't let you multitask programs, however, and it uses DOS as its I/O system. This month I'll show you how it works; next month I'll dissect the code itself.

This month, I'm also passing on a neat routine for computing a running mean, sent in by Kevin Jennings.

What is Multitasking?

Let's start with some definitions, just so everybody is starting from the same base.

An operating system is a collection of programs that lets you execute other programs. Typically, some of these programs are resident (they stay in the computer's memory), others are nonresident (they stay on the disk until they're needed, whereupon they're read into main memory and executed), and some are somewhere in between (they stay in memory until the space is needed for something else, whereupon they are overwritten, but they'll be read back into memory once space is available).

by Allen Holub

The main parts of the operating system are the kernel (or executive), the I/O system, and the shell. The kernel is in charge of actually reading and executing other programs; the I/O system takes care of all communications with the outside world—the disk, printers, terminals, and so forth; and the shell is the part that talks to you. The shell typically uses both the I/O system and the kernel. This month's column does not present an operating system. In fact it only looks at a small part of the kernel—the part that takes care of multitasking.

Multitasking is a method for making a computer appear to be doing several things at once. In reality, it's swapping back and forth between various tasks very quickly. Nonetheless, from a human user's perspective, several things seem to be going on at the same time. You can actually do multitasking without a kernel. Consider an interrupt-driven terminal multiplexer—a device that collects lines of text from several terminals and passes them (accompanied by some sort of identifying information) along one communications channel to a single mainframe computer. You might have eight input channels, each of which would trigger a unique interrupt in the processor. There would also be eight interrupt-service routines, each of which would collect characters until an end of line was found and then send that line to the mainframe. Each routine would maintain its own input buffer and attach a unique identifier to each line. The multiplexer would appear to be doing eight things at once, reading from eight input sources simultaneously, but in reality it would be performing eight independent tasks when stimulated by distinct external events (characters arriving). If eight characters were to arrive simultaneously, the multiplexer might fail because of insufficient time to react to all eight inputs before more characters come along.

A different approach to the same problem is to have a single interrupt, usually created by a timer chip of some sort, that starts up a scheduler subroutine. The scheduler activates a particular subroutine, just as if it had been activated by the interrupt itself. When the next timer interrupt comes along, the scheduler suspends the current subroutine and activates a new one, just as if a second, higher-priority interrupt had come along. The scheduler also takes care of some of the overhead involved with the interrupt service. In particular, it does a context swap—it saves all the registers (including the instruction pointer) and preserves the running subroutines's stack. (Each task has its own stack, just as a well-behaved interrupt-service routine should have its own stack). A task, then, is different from a normal subroutine in that it is activated only by the scheduler, not by a subroutine call in the common sense, and it has its own stack and a place to save registers during a swap.

Note that a task is not the same as a process in normal operating system parlance. A process usually implies a program that is read in from the disk and executed by the operating system. A task, however, is any independently executing code. In the case of the kernel presented here, a task is a subroutine (and the subroutines that it calls) that has its own stack and register set. That is, it is the stack and register-save area that define the task, not the code itself. Several tasks can share the same code—provided that

OUR NEW GRAPHICS SOFTW
MAY BE A BIT FASTER TH

ARE FOR dBASE AND R:BASE AN WHAT YOU'RE USED TO.

DB Graphics turns raw dBASE III/III PLUS and R:BASE data files into graphics with no conversion, export or import.

Up until now, turning the raw data of a dBASE III/III PLUS or R:BASE file into easily understood graphs and charts has been anything but quick.

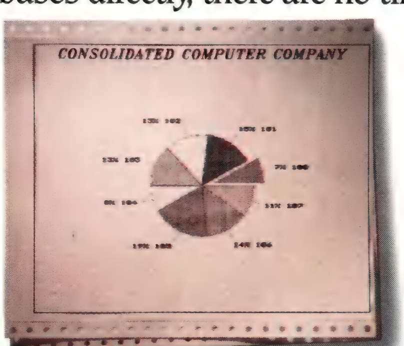
It's meant plodding back and forth between your database and your graphics program, through file conversions and translations, until you finally got a chart that made your point.

But now there's a faster route to arresting presentation graphics. Now there's DB Graphics.

THE FIRST GRAPHICS SOFTWARE THAT GOES STRAIGHT TO THE HEART OF YOUR DATABASE.

DB Graphics is the only presentation graphics software designed specifically for dBASE, R:BASE and other database software users.

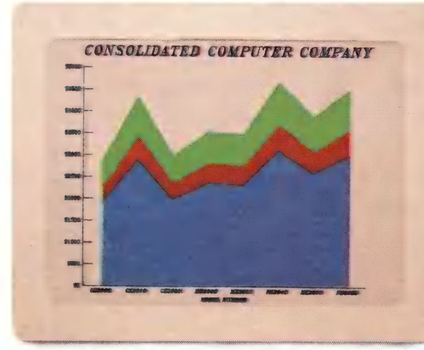
Since it reads dBASE and R:BASE databases directly, there are no time-consuming export, import or conversion utilities to fool with. DB Graphics can also sort and group fields and use conditional operators and other powerful database management tools to precisely select and graphically display the data you need. And because it has a direct line to your live data, you can modify a graph, or choose different data to represent, in an instant.



You can change the style of a graphic, or the data it represents in an instant.

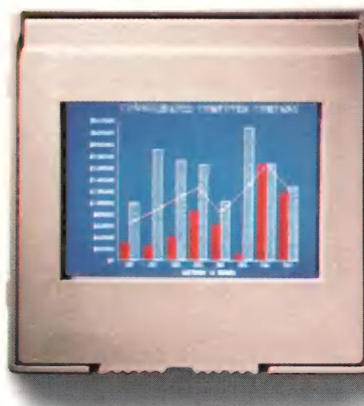
MAKE A GRAPHIC IMPROVEMENT IN ALL YOUR PRESENTATIONS.

Even though DB Graphics is more intelligent than other graphics programs, it's no less attractive. It allows you to display your data in eight different graph types—pie, bar, high/low, area, scatter, mixed, column, and line. You can work with 16 colors and hundreds of vivid combinations of textures and patterns. Plus, you can choose from seven type styles for free text and labels.



Powerful database management tools make it easy to get the data you need into a graph.

In fact, there's really only one kind of database presentation DB Graphics isn't capable of making. A dull one.

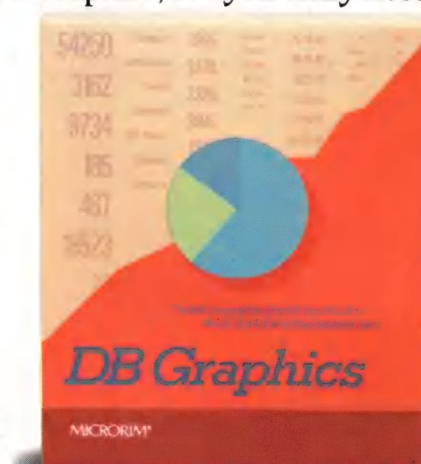


DB Graphics can produce a variety of output, including 35mm slides.

FOR A TRIAL PACK, CALL 1-800-624-0810 DEPT. DR1287.

We could go on about the power, speed and accuracy of DB Graphics, but you really need to experience it for yourself.

So call our toll-free number (or from Alaska or Canada call 1-206-867-1800 Dept. DR1287) and order a DB Graphics Trial Pack for just $9.95 today. And start making the most out of your database.



The only presentation graphics software specifically designed for dBASE, R:BASE and other database software users.

DB Graphics also works with Lotus 1-2-3, Symphony, Multiplan and other data management files after translating to the ASCII, dBASE III/III PLUS or R:BASE format. DB Graphics is compatible with the IBM PS/2, PC, XT, AT and 100% compatible personal computers. It supports the IBM VGA, MCGA, EGA, CGA and Hercules display standards. It also supports most popular plotters, graphic printers, camera systems and laser printers from IBM, Epson, Hewlett-Packard and many others.

CIRCLE 199 ON READER SERVICE CARD

C CHEST

(continued from page 126)

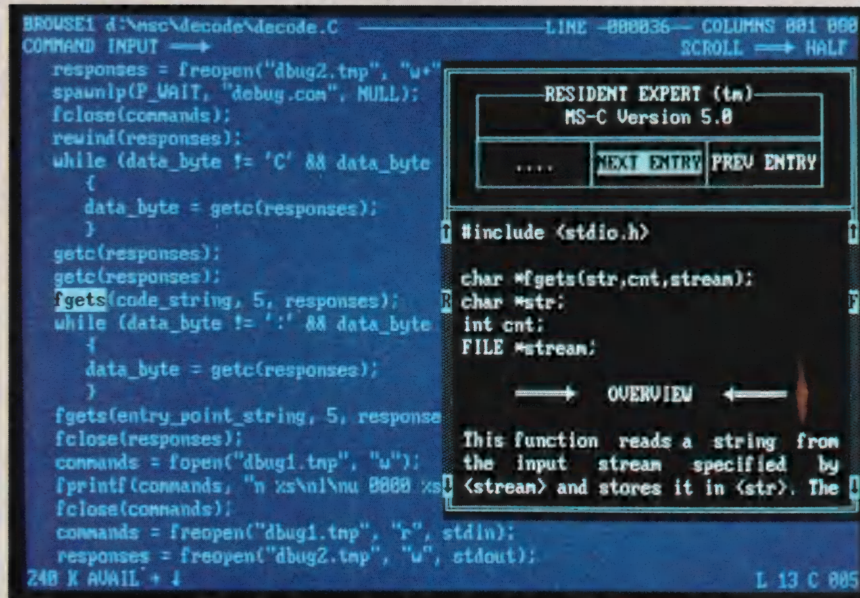
the code doesn't use static variables. Because each task has its own stack, and because the program counter is saved as part of the context swap, the local variables (on the stack) will be distinct and each task will remember where it was in the code when it's interrupted. Much of the context-swapping code assumes that everything is contained in a single, 8086, small-model program, however. You'll have to play with the program a little to make it support other models or external program execution.

Several strategies are used to determine which task gets control when the timer interrupt comes along. In round-robin scheduling, the various tasks are activated one at a time until all of them have been executed once, then the processor goes around the circle again. In a priority-based system, each task is assigned a unique priority. All nonrunning tasks are organized into an active list. When an interrupt comes along, the scheduler compares the priority of the running task with that of the highest-priority task in the active list and, if necessary, puts the running task to sleep and activates the top task in the list. It's possible to have a system that combines both of these strategies—tasks with the same priority are executed in round-robin fashion but a higher-priority task takes precedence over lower-priority ones.

There are two approaches to multitasking, too, that are useful in different situations. The method I've just described is preemptive multitasking. The timer interrupt causes the next task to preempt the current one. You can also have a nonpreemptive system (such as Microsoft Windows) in which individual tasks voluntarily give up control when they don't need the CPU anymore. There are two advantages to the nonpreemptive approach. You don't need a timer interrupt and CPU time is not wasted figuring out that the scheduler doesn't need to do a context swap. On the down side, a task retains control until it yields (gives up control). If the running task crashes, the whole system stops.

Because tasks aren't subroutines in the normal sense of the word, they can't pass information to each other using arguments and return values. Intertask communication is then done with a message-passing system. The operating system lets you create a set of message queues, or mailboxes, to which messages can be sent. Other tasks can then wait at the queue for a message to arrive. Several tasks can wait at the same queue—they're just given messages as they arrive, and each task gets one message. If no tasks are waiting, an incoming message is queued up until a task comes along to fetch it. A task that's waiting for a message is suspended—it will not be activated by the scheduler until a message arrives at the queue—though it's also possible for a task to time-out—the task is put back into the active list if the message hasn't arrived within a specified time. Once the message arrives, if the receiving task is of higher priority than the running task, it gets control immediately; otherwise, the task (and the message) are put back into the active list and will be reactivated in the normal way. Note that a mailbox is a data structure that contains two queues: a queue of

RESIDENT EXPERT Pop-up Reference Guides...



Try One And Get Our MS-DOS/PC-DOS Guide Absolutely FREE!

THE POP-UP REFERENCE REVOLUTION BEGINS

How much development time could you save if you never had to open another PC language or technical reference manual again? What if you could just point at a compiler keyword, assembly instruction, or function name *on your screen* and with a keystroke have complete, authoritative information about language syntax, operands, parameters, examples, and much more?

INTRODUCING THE RESIDENT EXPERT SYSTEM

A growing library of comprehensive, disk resident reference guides about the PC *and* your favorite PC languages. All available instantly through our unique memory resident pop-up access system.

VIRTUALLY EVERYTHING YOU NEED TO KNOW

Each of our *Compiler Reference Guides* contains virtually everything you need to know to program with your *preferred implementation* of your favorite language. Language syntax, all library functions, complier directives, and error codes are thoroughly documented.

Our *PC Programmer's Reference Guide* documents every PC (and AT) processor instruction and every BIOS and DOS service interrupt. You'll also find tables of keyboard codes, line drawing, ASCII, and IBM character sets, and much more.

THE SPECIALIST'S LIBRARY

Your compiler is unique. That's why our reference guides are *specialized...*each one designed for a particular vendor's language implementation.

Free With Any Purchase!

Our Companion Pop-up Guide To MS-DOS 3.2/PC-DOS 3.3

Limited Time Offer

QUICK DRAW ACCESS SYSTEM

*Point-and-shoot...*just place the cursor over any term on your screen. Chances are we've got it fully detailed in one of our data bases.

*Fully cross indexed...*if the instruction or library function you're using isn't quite right, our related topics cross index can help you find a better one.

*Multiple volumes on line...*you can have one or a dozen of our pop-up reference guides on line...a complete library available instantly.

THE INFORMATION YOU NEED...WHERE YOU NEED IT

Our pop-up shell varies its size and shape dynamically, only taking as much space on your screen as it needs and it *never* covers your working area. You can see your work and our reference data at the *same* time.

A COMPLETE LIBRARY...STILL ONLY A BEGINNING

At Santa Rita, our commitment is to provide the most accurate, extensive selection of PC language reference materials available. If you don't see one of our guides for your favorite language or compiler listed below don't worry, we're probably working on it!

PC Programmer's Reference Guide . . . $59.00
(with Assembly Language Guide)

Borland Turbo C *(1.0)* 59.00
Borland Turbo Pascal *(3.0 and below)* 59.00
(with Graphics & Numerical Methods Toolbox)
Borland Turbo Prolog *(1.1 and below)* 59.00
(with Prolog Toolbox)
Lattice C Compiler *(3.2 and below)* 59.00
Mark Williams LetsC *(4.0 and below)* 59.00
Microsoft C Compiler *(5.0 and below)* 59.00

SantaRita

For the location of your nearest Santa Rita Software dealer, or to order direct, call us at 1-214-727-9217. We'd like to hear from you.

Santa Rita Software
1000 E. 14th Street, Suite 365
Plano, Texas 75074

The RESIDENT EXPERT System

CIRCLE 201 ON READER SERVICE CARD

(continued from page 130)

messages and a queue of waiting tasks.

A problem with this communications method is deadlock, a situation in which two tasks are both waiting for messages that can only be sent by the other task. That is, task A is waiting for a message that can only be sent by task B, and at the same time, task B is waiting for a message that can only be sent by task A. A similar situation can arise when every task in the system is waiting for a message simultaneously.

Another problem is resource blocking. The DOS I/O system is a good example of why this blocking is necessary. DOS is not reentrant, which means that a DOS function cannot be called reliably from an interrupt-service routine. Put another way, once you call DOS, you can't call DOS a second time until the first DOS call has finished. Because task activation can happen at any time, even when you're inside DOS, some method is needed to turn off the scheduler when a task is doing a DOS call. A task blocks to get control of the CPU—no other task will be activated as long as scheduling is blocked. Note that the system will halt if scheduling is blocked and the running task crashes.

An alternate approach to blocking that doesn't have this disadvantage is an exclusion semaphore. An exclusion semaphore is just a queue of length 1. A message is waiting at the queue when a resource (such as DOS) is available. To use the resource, a task gets the message, uses the resource, and then reposts the message to free up the resource. This approach has its disadvantages, too. If the task that has the resource is suspended for some reason, no other task can use the resource until the task is reactivated.

Note that, though you can block by disabling interrupts, that's not usually a good idea. First of all, on the IBM PC you'll mess up the system clock, which is interrupt-driven. The other problem is the console I/O and disk subsystems, which are also interrupt-driven. You won't be able to send or receive characters when interrupts are off. You're also likely to mess up your disk transfers. If you do disable interrupts, do it for the shortest time possible. Also note that tasks must be treated as interrupt-service routines. In practice this means that you must always block before you call DOS.

Kernel Users' Manual

This section is just a users' manual for the kernel subroutines. I haven't gone into any implementation details, all of which I'll discuss next month.

Several error codes are declared in kernel.h and are returned by the various multitasking subroutines. The codes are shown in Table 1, below. *TE_NOERR* has the value 0, and the other error codes are small negative numbers. In addition, kernel.h holds *typedefs* for the TCB and *T_QUEUE*. The former is a task control block, used to hold a task's stack and so forth; the latter is a message queue. I'll discuss all of these codes in greater depth later.

Various global variables and subroutines are used internally. These are listed in Table 2, below. You shouldn't use the names in your own programs, because, for the most part, these variables are not useful to an application program. The possible exception is *T_clock*, the system clock. *T_clock* is incremented on every system clock tick. It is not incremented while scheduling is blocked, however. Because *T_clock* is an *unsigned long*, rollover is not a problem. If you assume the default of 18.2 ticks/second, the clock will roll over after about 65,552 hours (about 7.47 years):

((0xffffffff/18.2)/60)/60 = 65,552 hours
65,552/24/365.35 = 7.47798 years

Of course, this number will scale with faster tick rates, but the resolution should be OK for all reasonable tick rates. The other variable of possible interest is *T_numtasks*, which holds the number of tasks that have been created so far.

There are a few additional considerations. The Microsoft compiler inserts a call to a subroutine called *_chkstk( )* at the head of every subroutine. This can cause problems because a task is running on its own stack, not on the one that *_chkstk( )* expects it to be using. The problem is solved by *t_start( )*, which replaces the default *_chkstk( )* with its own version. The new version checks the current stack pointer against the base of the task's local stack. If a stack overflow happens, multitasking is terminated and *t_start( )* will return an error code (see later).

Starting and Stopping Multitasking

A typical program will first create a few tasks (I'll look at how in a moment) and then start up the multitasking environment. (Tasks can

TE_NOERR	No error
TE_TOOMANY	Maximum number of tasks (32) already exists
TE_NOMEM	Insufficient memory available
TE_BADARG	Illegal argument
TE_TIMEOUT	Time-out
TE_QFULL	Queue is full
TE_NOTASKS	No tasks to send message
TE_INTERNAL	Internal error
TE_DEADLOCK	Delete would have caused a deadlock
TE_STACK	Stack overflow
TE_KILL	Ctrl-Break encountered

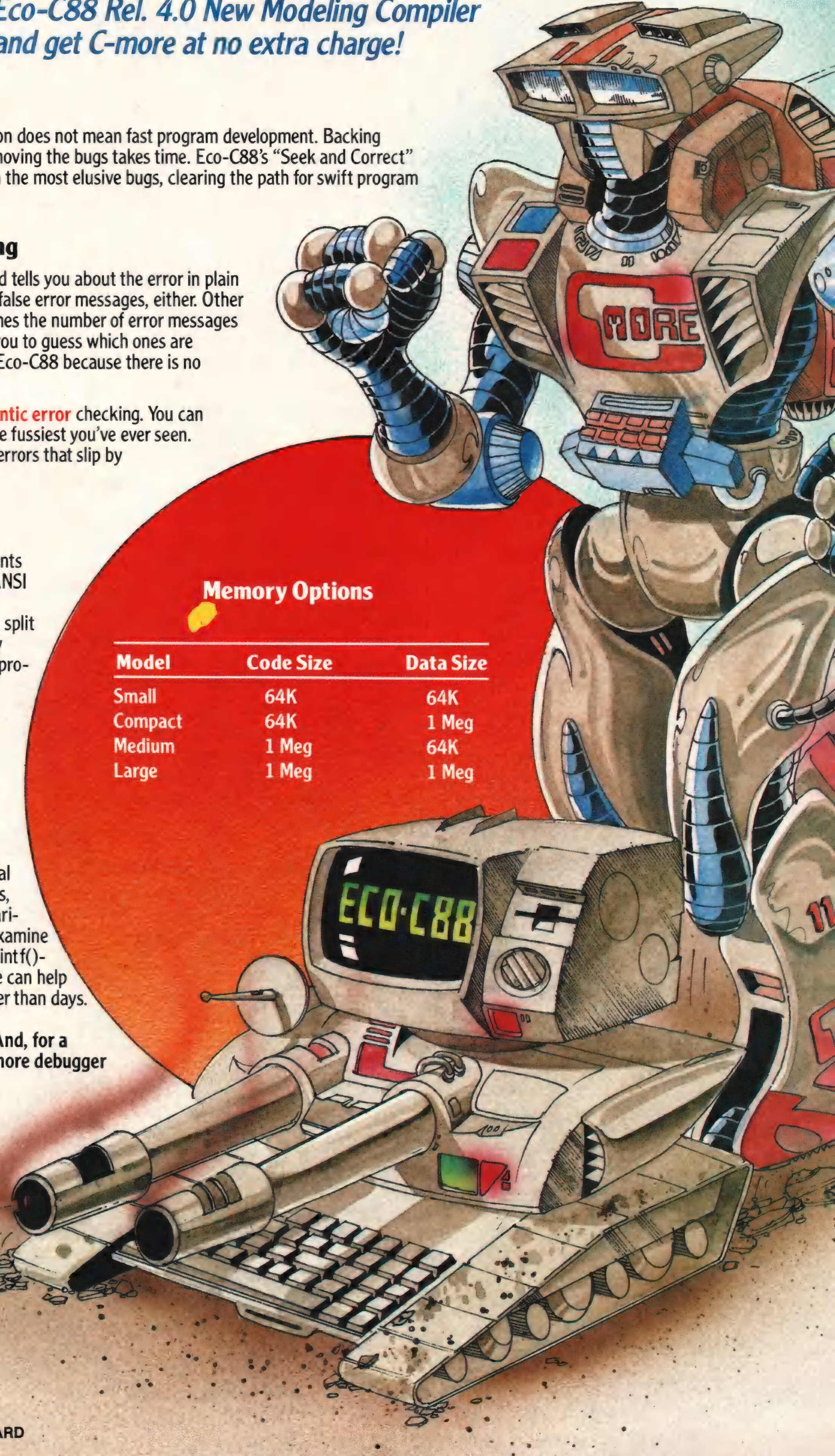
Table 1: *Error codes declared in kernel.h*

TCB	*T_active;
unsigned long	T_clock;
int	T_numtasks;
T_QUEUE	*T_queues;
PQ	*T_tasks;
void	_t_reschedule( );
void	_t_install( );
void	_t_swap_in( );
void	_t_shazam( );
void	_t_speedup( );
void	_t_slowdown( );

Table 2: *Global variables and subroutines used by kernel.h*

DEBUGGING SWAT TEAM

Order Eco-C88 Rel. 4.0 New Modeling Compiler and get C-more at no extra charge!



Seek and Correct

You already know that fast compilation does not mean fast program development. Backing up for bogus error messages and removing the bugs takes time. Eco-C88's "Seek and Correct" three - way error checking finds even the most elusive bugs, clearing the path for swift program development.

Double Barrel Error Checking

Eco-C88 nails syntax errors cold and tells you about the error in plain English. And there's no avalanche of false error messages, either. Other compilers can generate up to four times the number of error messages actually present; they leave it up to you to guess which ones are real. You'll be more productive with Eco-C88 because there is no guess work.

Eco-C88 provides ten levels of semantic error checking. You can select from almost no checking to the fussiest you've ever seen. Eco-C88's "picky flag" finds subtle errors that slip by other compilers.

Eco-C88 also features:

- All data types, plus ANSI Enhancements
- Robust library, including many new ANSI functions
- CED editor with online function help, split windows, compile-edit-link capability
- New, expanded manual with sample programs for the library functions

Memory Options

Model	Code Size	Data Size
Small	64K	64K
Compact	64K	1 Meg
Medium	1 Meg	64K
Large	1 Meg	1 Meg

C-more Source Code Debugger

Finally, if a really nasty bug persists, put C-more, our source code debugger, to work. With C-more you can watch your program as it executes, single-step it, set simple or conditional breakpoints, test complex expressions, use variables as indexes into other variables, initialize and trace variables, examine CPU registers, display results with printf()-type options and much more. C-more can help you track down bugs in minutes rather than days.

The price for Eco-C88 is $99.95. And, for a limited time, we'll give you our C-more debugger at no extra charge.

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

(317) 255-6476 (Tech Info)
(800) 952-0472 (Orders)

(continued from page 132)

create other tasks, too. They don't all have to exist before multitasking is started.) Multitasking is started with a call to:

```
int t_start( speedup )
int speedup
```

Speedup is a system-clock speedup factor. If it's 0, then the system will not be preemptive and tasks will have to yield (either with a call to *t_yield()* or by waiting for a message) to give up control of the CPU. A speedup factor of 1 uses the default PC clock rate (roughly 18.2 interrupts/second). A factor of 2 is twice as fast (36.4 interrupts/second). If the factor is too large, the system will actually slow down because it will start missing interrupts. It's best if the speedup factor is a power of 2.

At least one task must have been created prior to the *t_start()* call. Control passes immediately to the highest-priority task. *T_start()* returns either when all tasks are deleted or when *t_stop()* (discussed later) is called. *TE_NOTASKS* is returned if no tasks exist initially—multitasking is not started in this situation. *TE_NOERR* is returned when all tasks have been deleted successfully. No tasks can be waiting on queues. This is the normal way to return. *TE_STACK* is returned as soon as a stack overflow in any task is detected. *T_active* will point at the TCB of the offending task.

TE_DEADLOCK is returned when the only active task in the system deletes itself. Other tasks exist but they're all pending on queues. That is, there must always be at least one running task in the system. To accomplish this, a very-low-priority idle task is often created. This task doesn't do anything but spin around; it's only active when all other tasks are doing something else, and it deletes itself when it's the only task left in the system. A typical idle task is shown in Example 1, below.

If *TE_NOERR* is returned, then all memory allocated to the tasks will have been restored to the heap; otherwise, if one of the above errors was returned, *T_active* will point at the TCB of the offending task. Other return values are possible if a task calls *t_stop()* directly (see later).

A panic abort from the multitasking environment can be accomplished with a call to:

```
t_stop( errcode )
int errcode;
```

Multitasking is turned off, and control passes back to the routine that called *t_start()* (immediately following the *t_start()* call). That is, *t_stop()* forces *t_start()* to return—it does not itself return. *Errcode* is passed back to the calling routine as the return value of *t_start()*. The process is analogous to a Unix *exit()* call, which doesn't return and whose argument is passed back to a *wait()* call in the parent process.

Creating and Deleting Tasks

Tasks are created with a call to:

```
TCB *t_create( subr, tag, priority,
                    stack_size, ..., NULL)

int (*subr)( );
char *tag;
unsigned priority;
int stack_size;
```

Subr is a pointer to the subroutine that forms the main module for the task. *Tag* is a string used to identify the TCB—it's used only for debugging. *Priority* is the task's priority—the higher the number, the higher the priority. Priorities should be in the range 0–255. If more than one task has the same priority, the tasks are executed in a round-robin fashion. *Stack_size* is the stack size (in 16-bit words) for this task only. Note that a few Microsoft functions (such as *printf()*) use up inordinate amounts of stack. If you're going to call Microsoft library routines, you'll need at least 1K stacks (*stack_size* of 512).

The remaining arguments are a *NULL*-terminated list of pointer-size arguments that will be passed to the task at start-up. (More on this in a moment.)

T_start() forces a reschedule. That is, if the task that you're creating is of higher priority than the running task, the new task will get control immediately.

A pointer to the created TCB is returned normally. This pointer is useful if you want to delete the task later. Error return values are *TE_TOOMANY* (maximum number of tasks [32] already exists) and *TE_NOMEM* (insufficient memory available to create task).

An example of task creation is shown in Example 2, page 137. Here, a single task called *foo()* is created in *main()*. It is of priority 10 and has a 512-word (1K) stack. The remaining arguments are passed to *foo()* when the task starts up. *Foo()* will print its arguments and delete itself. (It prints "hello world.")

Note that a task should never return in the normal way. It should always delete itself rather than returning. If a return is executed (either explicitly or implicitly by falling off the bottom of the subroutine), then *t_stop()* is called immediately (with a garbage argument).

A task's priority can be changed at any time with a call to:

```
int t_chg_priority( tp, new_priority )
```

```
idle(\sc128\)
{
  int t, i ;

  do {
    for( i = 1000; --i>=0 ; )
      ;
    t_cli(\sc128\);          /* Clear interrupts */
    t = T_numtasks;          /* t = # of tasks   */
    t_sti(\sc128\);          /* restore ints.    */
  }
  while( t > 1 ) ;

  t_delete( NULL );      /* delete self      */
}
```

Example 1: *A typical idle task*

If you ever wanted to take a crack at assembly language, now's the time.

You probably already know that assembly language subroutines are the smartest way to get the fastest programs.

But if the complexities of working in assembler made you think twice, here's some good news. We've made Microsoft® Macro Assembler Version 5.0 a lot easier to use.

We eased the learning process by giving you the best support around. We completely revised our documentation. The new Mixed Language Programming Guide gives you step by step instructions for linking your assembly code with Microsoft QuickBASIC, C, FORTRAN, Pascal and other languages. And you get a comprehensive reference manual with listings of the instruction set and examples of each instruction. We didn't stop there, though. You also get an on-disk collection of templates and examples.

We've also dramatically simplified the high-level language interface. In just a few simple steps, you can be calling Macro Assembler subroutines from programs written in your favorite language.

Now that you're writing the fastest programs, Microsoft is giving you the fastest way to debug them. For the first time, we've added our CodeView® debugger to Macro Assembler.

With source code and comments on your screen, Microsoft CodeView makes debugging programs containing assembly language subroutines a snap.

And you'll be glad to know that you don't sacrifice any speed for all the ease of use. We took the fastest Macro Assembler on the market and made it even faster.

So what are you waiting for? Get your hands on Microsoft Macro Assembler and see what it's like to break your personal speed limit.

Microsoft®

For more information or for the name of your nearest Microsoft dealer, call (800) 426-9400. In Washington State and Alaska, (206) 882-8088. In Canada, call (416) 673-7638.


THE CURE
FOR COMMON CODE
Are you getting the recommended monthly allowance of C, Assembly, Forth, Pascal, BASIC or Modula-2? Subscribe to *Dr. Dobb's Journal of Software Tools* and you won't catch any nasty bugs again!
Each month the Doctor brings you aid for ailing algorithms and the cure for common code. For the latest developments in software design and pages of code that will make you a more productive programmer, take the Dr. Dobb's prescription.
For more than a decade, the programming elite have known *Dr. Dobb's Journal* to be the foremost source of software tools. Subscribe now and get your monthly dose from the Doctor.
SINCE 1976
Dr. Dobb's Journal
INGREDIENTS
Articles, Reviews, C Chest, Software Toolbox, Swaines Flames, Structured Programming, Artificial Intelligence, DDJ Online, The Examining Room, Programming Paradigms
PASCAL
PROLOG
FOR
C
BASIC
ASSEMBLY
MODULA-2

SUBSCRIBE AND SAVE!

$5 SAVINGS

Subscribe to

DR. DOBB'S JOURNAL OF SOFTWARE TOOLS

and save over $5—a 15% savings off the cover price!

☐ 1 year $29.97 ☐ 2 years $56.97

☐ Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______ Exp. date ______

Signature ______

Name ______

Address ______

City ______ State ______ Zip ______

ONLY $29.97! YOU SAVE OVER $5.00

Savings based on a full one-year cover price of $35.40. Canada & Mexico add $10 for surface mail per year. All other countries add $27 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. **3465**

COMMENTS & SUGGESTIONS

December 1987, #134

Dear Reader,

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take time to write a letter. This card provides you with an easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail.

—Ed

Which articles or departments did you enjoy the most this month? Why?

______

______

______

Comments or suggestions ______

______

______

______

Name: ______

Address: ______

______

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
Box 3713
Escondidio, CA 92025-9843

C CHEST
(continued from page 134)

```
TCB *tp;
int new_priority;
```

Like *t_create()*, this routine forces a reschedule. If the task was waiting on a message, it is immediately timed-out and put back onto the active list. A task may change its own priority. This routine returns *TE_NOERR* normally and *TE_BADARG* if the task doesn't exist or if the priority of the task is greater than 255.

Tasks are deleted with a call to:

```
int t_delete( task )
TCB *task;
```

which also frees all memory associated with *task*. Note that memory that the task itself allocates (via a *malloc()* call or equivalent) is not freed—only the memory that *t_create()* allocated (for the stack and so forth) is released. A task may delete itself with a *t_delete(NULL)* call. *T_delete()* forces a reschedule when the current task is deleted. Return values are *TE_NOERR* on success and *TE_BADARG* if the task doesn't exist.

Messages

Three routines are used to create message queues (mailboxes) and for passing messages between tasks. Queues are created with a call to:

```
T_QUEUE *t_makequeue( size )
int size;
```

Size is the maximum number of messages that can be waiting in the queue. Any number of tasks can wait at a queue, however. Normally a pointer to the queue is returned, but *TE_NOMEM* is returned if there's insufficient memory. The queues are linked into a linear list that is searched for timed-out tasks on every system clock tick, so it's best to create the most active queues first.

Messages are sent from a task to a queue with a call to:

```
int t_send( q, msg )
T_QUEUE *q;
void *msg;
```

where *q* is a pointer returned from a previous *t_makequeue()* call and *msg* is a pointer to the message. The pointer, not the message, is stored, so the messages can be anything you want. Typically, *msg* will be a pointer to a structure or a string. The message itself must be in static memory.

If no tasks are waiting at the queue, *t_send()* returns immediately; otherwise, the message is attached to the task and the scheduler is called. This means that if the waiting task is of higher priority, control will be taken away from the task that sent the message and given to the waiting task. *TE_NOERR* is returned normally. Error returns are *TE_BADARG* on a bad *q* argument and *TE_QFULL* if the queue is full.

The other side of the message-passing system is:

```
void *t_wait( q, timeout )
T_QUEUE *q;
int timeout;
```

which is used by a task to wait at a queue for a message to arrive. *Q* is a pointer returned from a previous *t_makequeue()* call and *timeout* is a time-out value. The task will only wait for *timeout* system clock ticks before it is put back onto the active list. (Remember, though, the clock

```
foo( a, b )
char *a, *b;
{
  t_printf("%s %s\n", a, b );
  t_delete( NULL );
}

main(\sc128\)
{
  t_create( foo,"foo", 10, 512, "hello", "world", NULL);
  t_start( 2 );
}
```

Example 2: *An example of task creation*

C CHEST
(continued from page 137)

won't tick when the scheduler is blocked). The maximum time-out is 32,767 clock ticks (that's about half an hour at 18.2 ticks/second). If the time-out is 0, *t_wait()* returns immediately (without a reschedule) if no messages are waiting in the queue.

Message requests are queued up in order received, without regard to priority. I've done this both because it's easy and because, in most applications, tasks with different priorities will not be pending on the same queue.

If a message is present in the queue, *t_wait()* immediately returns the pointer to the message (the same pointer as was passed into *t_send()*) without a reschedule; otherwise, the current task is suspended (removed from the active list) until a message arrives.

Normally, a message pointer is returned. Error return values are *TE_TIMEOUT* (on a time-out or if the *timeout* argument is 0 and no message is waiting) and *TE_NOTASKS* (if the current task is the only running task—a guaranteed deadlock).

Blocking and Yielding

Four subroutines are provided to support blocking: *void t_cli();*, *void t_sti();*, *void t_block();*, and *void t_release();*.

T_cli() and *t_sti()* disable and enable interrupts. Use these advisedly. Interrupts should never be off for extended periods. They should never be off when you're using the DOS I/O functions.

T_block() and *t_release()* are more reliable. *T_block()* disables the scheduler but not the normal clock interrupt. That is, the current task retains control of the system once *t_block()* is called. Because interrupts are not disabled, this is a much safer routine to use than *t_cli()*. There is one caveat, however. If a normal interrupt happens while interrupts are disabled, the hardware will execute the interrupts as soon as they're enabled again. This is not the case with *t_block()*. That is, the scheduler does not know whether an interrupt happened while it was blocked. Consequently, a tight loop such as this:

```
while( *str )
{
    t_block( );
    putchar( *str++ );
    t_restore( );
}
```

won't work as expected. Because so little time elapses between the *t_restore()* and the next *t_block()* call, the odds are that an interrupt will never occur while the scheduler is active. That is, the following will probably work in just the same way as the previous example:

```
t_block( );
while( *str )
    putchar( *str++ );
t_restore( );
```

Note that a *t_cli()* can disrupt the DOS system clock (time will be lost) while interrupts are off; *t_block()* doesn't have this limitation.

The final control-related subroutine is *void t_yield();*. This routine puts the current task to sleep and activates the task that has the next-highest priority. Note that the current task is always suspended, even if it's the highest-priority task. It will get back control on the next timer interrupt in this case, however. *T_yield()* is used primarily when the scheduler is blocked or when you're running a nonpreemptive system. It's also useful when a high-priority task doesn't do much and doesn't want to hog the CPU. *T_yield* returns *TE_NOERR* when

```
#include <stdio.h>
#include <tools/video.h>
#include "kernel.h"

#define TEST 3 /* 1 = nonpreemptive test,
                * 2 = preemptive test: simple timer
                * 3 = test round-robin scheduling
                */

T_QUEUE *Queue1;
T_QUEUE *Queue2;

main(\sc128\)
{
  int status = 0;
  long t_numint(\sc128\), t_numblk(\sc128\);
  int  sam(\sc128\), dave(\sc128\), timer(\sc128\), idle(\sc128\),
                                                        maintask;

  if( !(Queue1 = (T_QUEUE *) t_makequeue( 2 ) ) )
      printf("Can't make Queue1 queue\n"), exit(1);

  if( !(Queue2 = (T_QUEUE *) t_makequeue( 2 ) ) )
      printf("Can't make Queue2 queue\n"), exit(1);

 #if (TEST == 1)
  status = (int) t_create(sam, "sam", 100, 512, "SAM", NULL);
  status = (int) t_create(dave, "dave", 50, 512, "DAVE", NULL);
  status = t_start( 0 );
 #endif

 #if (TEST == 2)
  status = (int) t_create(timer,"timer",10, 512,"timer",NULL);
  status = (int) t_create(idle, "idle", 1, 100,    NULL);
  status = t_start( 2 );
 #endif

 #if (TEST == 3)
  status = (int) t_create(maintask,"maintask",200, 512, NULL);
  status = t_start( 2 );
 #endif

  t_perror( "\ndone: ", status );
  t_sstats(\sc128\);
  printf("%ld interrupts, %ld blocked\n", t_numint(\sc128\),
                                              t_numblk(\sc12;
}
```

Example 3: *A program that creates queues, starts multitasking, and performs three tests*

the yield was successful and *TE_NOTASKS* if there were no tasks to which to give control.

Note that *t_yield()* normally doesn't return until after it regains control of the system. That is, the normal control flow goes like this:

1. A task calls *t_yield()*.
2. The second task is activated.
3. The second task is suspended.
4. Control goes back to the original task and *t_yield* returns *TE_NOERR*.

Statistics

Several debugging and statistics routines are provided. Two routines are useful for debugging:

```
t_tprint( t )
TCB *t;

t_qprint( q )
T_QUEUE *q ;
```

T_tprint() is passed a TCB pointer, and it prints the TCB in human-readable form to standard output; *t_qprint()* does the same for the queues.

Several statistics routines are also useful: *long t_numint();* returns the number of unblocked interrupts, and *long t_numblk();* returns the number of blocked interrupts. Finally, *t_sstats()* prints various scheduler-related statistics (the number of times the scheduler was called, the number of time-outs, and the number of context swaps done by the scheduler). This routine should not be called from a task (because it uses *printf()*).

Miscellany

int t_second()
Returns the number of system clock ticks in a second, given the speedup factor passed to *t_start()*. It returns 0 if the speedup factor was 0. In this case, a *t_wait()* call will never time-out.

*char *t_errlist[]*
Works like the normal *errlist[]* that's supported by most C compilers. Indexed by error code, it evaluates to a string that holds an appropriate error message.

t_iserr(x)
Returns true if *x* is an error code and false otherwise.

t_perror(str, errcode)
*char *str;*
Prints an error message. If *errcode* is an error code, it prints an appropriate message; otherwise, it prints *status XXX*, where *XXX* is *errcode* represented as a decimal number.

t_printf(fmt, ...)
*char *fmt;*
Works like *printf()* except that it blocks before printing anything.

T_PRIORITY(a,b)
*TCB *a, *b;*
This macro, which is in kernel.h, compares the priorities of the two tasks. It returns a negative number if *a*'s priority is less than *b*'s, 0 if the priorities are the same, and a positive value if task *b* is of higher priority than task *a*. Note that a task's priority takes into consideration both the priority passed to *t_create()* and the time at which the task was last suspended. That is,

```
dave( arg )
char  *arg;
{
  char *s;

  t_printf( "In dave(%s), about to wait for message\n", arg );

  if( t_iserr(s = t_wait(Queue1, 100)) )
    t_perror("dave: first wait call", (int) s );
  else
    t_printf( "dave: got <%s> from Queue1\n", s );

  if( t_iserr(s = t_wait(Queue1, 100)) )
    t_perror("dave: first wait call", (int) s );
  else
    t_printf( "dave: got %s from Queue1\n", s );

  t_printf("dave: yielding\n");
  t_yield(\sc128\);

  t_printf("dave: deleting self\n");
  t_delete( NULL );
}
```

Example 4: *The task sam( )*

```
sam( arg )
char  *arg;
{
  int err;

  t_printf( "In sam(%s), yielding: no messages\n", arg);

  t_yield(\sc128\);

  t_printf( "sam: back from yield, sending messages\n");

  if( err = t_send( Queue1, "1st message") )
    t_perror( "Foo: sending 1st message", err );

  if( err = t_send( Queue1, "2nd message") )
    t_perror( "Foo: sending 2nd message", err );

  if( err = t_send( Queue1, "3rd message") )
    t_perror( "Foo: sending 3rd message", err );

  t_printf("Foo: yielding again\n");
  t_yield(\sc128\);
  t_printf("Foo: Returned from 2nd yield, deleting self\n");
  t_delete( NULL );
}
```

Example 5: *The task dave( )*

```
In sam(SAM), yielding: no messages
In dave(DAVE), about to wait for message
sam: back from yield, sending messages
dave: got <1st message> from Queue1
dave: got 2nd message from Queue1
dave: yielding
Foo: yielding again
dave: deleting self
Foo: Returned from 2nd yield, deleting self
done: No error

Scheduler called 0 times: 0 tasks timed-out, 0 context swaps
0 interrupts, 0 blocked
```

Example 6: *Output from test 1 Example 3*

C CHEST
(continued from page 139)

if two explicit priorities are the same, then the task that was suspended most recently is considered to be of lower priority than the earlier task. This mechanism makes round-robin scheduling easy to implement because tasks will move to the bottom of the list as they're suspended.

Some Examples

Let's look at a few examples. First, *main( )* must create a couple of queues and start up multitasking. I'll use a common *main( )* module for all the examples, which are shown in Example 3, page 138. One of three tests is performed, depending on the value of the *TEST* macro. Two queues, both of length 2, are created at the top of the subroutine. Then a few tasks are created and multitasking is started up with a *t_start( )* call. Finally, after control returns from *t_start*, various statistics are printed.

Now let's look at the individual tests. Test 1 exercises the nonpreemptive mode and demonstrates message passing. The two tasks, *sam( )* and *dave( )*, are shown in Examples 4 and 5, left. They both print their start-up arguments (from the original *t_start( )* call) and then pass several messages back and forth. Note that, though I'm passing strings as the messages, you could pass pointers to anything. Both the *t_wait( )* calls and the *t_yield( )* calls cause control to be passed between tasks. Also, *sam( )* intentionally tries to enqueue too many messages to see if the error mechanism is working properly. The output from the program is shown in Example 6, left. Note that all the statistics at the bottom of the output are 0 because the system isn't preemptive.

The next test is a small timer. It uses the *idle( )* task that I looked at earlier. The *timer( )* task is shown in Example 7, page 141. It prints a start-up message and then enters a *for* loop that executes five times. The *t_wait( )* call times-out after one second because nobody's sending any message to the queue. Consequently, the timer will print five ex-

clamation points, one every second, and then delete itself.

The third test is shown in Example 8, below. This test demonstrates several things. First of all, *main()* creates only one task, *maintask()*, which in turn creates two more tasks. Note, however, that both of these tasks use the same code! This is possible because every task has its own stack. Several tasks can share the same code, provided that they don't use static variables (the local variables will be on physically distinct stacks). Here, the task identifies itself by looking at its argument and it prints that argument every so often. The *dv_putchar()* subroutine is a direct-video output function, any of these functions will do. I generally use direct video in multitasking applications because going through DOS is so unpredictable (it messes with the interrupt system). Because the two tasks are of the same priority, they'll be executed in a round-robin fashion—five seconds of alternating 1s and 2s will be printed on the screen.

So that's how to use the subroutines. I'll look in depth at how they work next month.

Meaner Than Ever

Kevin Jennings writes:

"I read with some interest your column in the May 1987 issue of *DDJ* regarding the calculation of sample means. There is indeed a simple algorithm for computing the true mean of a set of data points on the fly. I've included both a block diagram [Figure 1, page 142] as well as a C function that implements it [Example 9, page 142—*I've taken the liberty of making Kevin's code a little more efficient*—Allen]. Call *mean()* with *reset* true the first time; thereafter, call it with *reset* false and *data* holding the current sample. The subroutine returns the running mean.

"The data comes in, one sample at a time, and is represented in the block diagram at Z(t). The algorithm computes a gain to apply to the difference between this measurement and what the filter thinks the measurement should be. This gain [K(t)] is then added to the previous estimate to give a new estimate. If equation 2 is rearranged as:

$$xhat(t) = (1-K(t))xhat(t-1) + K(t)Z(t) \quad (3)$$

and you look at what K(t) does as the samples come in, you should be able to convince yourself that this algorithm does indeed return the sample mean of all data received from the time that the filter is reset up to the current measurement. I don't recommend that you actually implement equation 3, however, because it's inherently less accurate

```
timer(\scl28\)
{
  int i;

  t_printf("Starting up timer\n");

  for( i = 5; --i >= 0 ; )
  {
    t_printf( "!" );
    t_wait( Queue2, t_second(\scl28\) );
  }

  t_printf("Deleting timer\n");
  t_delete( NULL );
}
#endif
```

Example 7: *The* timer( ) *task*

```
timer( arg )
{
  /* I'm assuming that pointers & ints are
   * the same size here.
   */


  int i;

  while( 1 )
  {
    for( i = 10000; --i >= 0 ; )
        ;
    t_cli(\scl28\);
    dv_putchar( arg + '0' );
    t_sti(\scl28\);
  }
}


maintask(\scl28\)
{
  TCB *t1, *t2;

  t1 = t_create( timer, "1st timer", 100, 512, (void*)1, NULL);
  t2 = t_create( timer, "2nd timer", 100, 512, (void*)2, NULL);

  t_perror( "task1:", (int)t1 );
  t_perror( "task2:", (int)t2 );

  t_wait( Queue2, 5 * t_second(\scl28\) );

  t_delete( t1 );
  t_delete( t2 );
  t_delete( NULL );
}
```

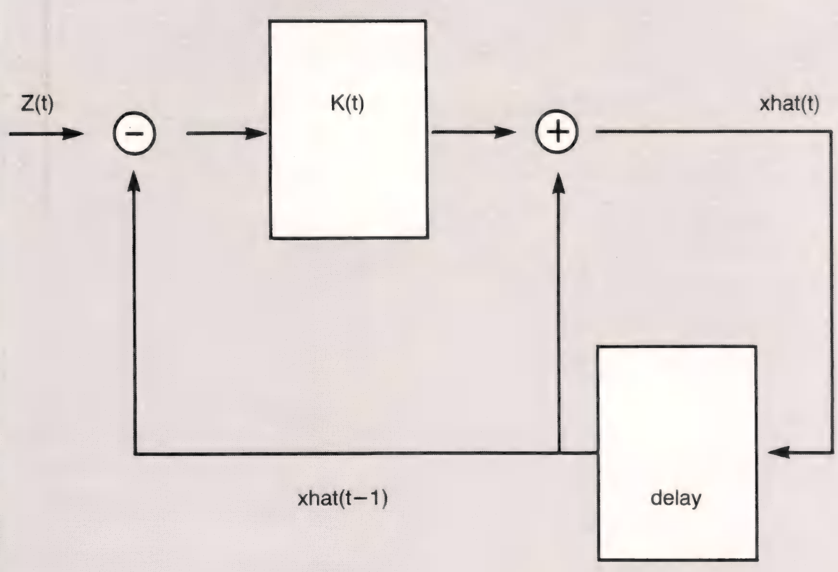
Example 8: *Test 3, used in Example 3*

(continued from page 141)

than is 2. Statisticians will recognize this algorithm as a least-squares estimator. It's also called a Kalman filter in digital signal processing.

"Note that it's more convenient to update the reciprocal of the gain [1/K(t)] than to update K(t) directly. Because the reciprocal takes on only positive integer values, it's tempting to declare *ki* as *unsigned int*, but then *ki* would wrap to 0 when it reached the maximum value for *unsigned int* (65,535, given a 16-bit *int*). You'd have to correct for the wraparound by holding *ki* at its maximum value once it's reached. Thereafter, the algorithm would no longer give you the exact mean, but no other algorithm would either. You're just trying to process too much data. I snuck around this difficulty by declaring *ki* as *double*. (*Double*s typically don't wrap when they overflow). In any event, because you have to do a floating-point divide to calculate the mean, there's no advantage in storing *ki* in an *unsigned int* because the compiler would have to convert it to *double* to do the arithmetic."

Kevin doesn't point out that the algorithm as implemented requires several floating-point operations, so it's slower than the exponential smoothing algorithm I presented in May, which used nothing but shifts and addition. On the other hand, the Kalman filter has an appealing simplicity and is more appropriate than exponential smoothing in many applications. It's certainly more accurate for small numbers of samples. Don't be tempted, by the way, to change the *double*s into *floats*. Most C compilers convert all *floats* to *double*s whenever they're used in an expression. Consequently, it takes longer to multiply two *floats* than it does to multiply two *double*s because you have to convert them to *double*s, do the arithmetic, and then truncate back down to *floats* to store the result. Moreover, any space savings that you get by using *floats* are usually lost in the additional code needed to do the type conversion.



$$1/K(t) = 1 + 1/K(t-1) \quad : 1/K(-1) = 0 \quad (1)$$
$$xhat(t) = xhat(t-1) + K(t)(\, Z(t) - xhat(t-1)) \quad : xhat(-1) = 0 \quad (2)$$

Figure 1: *Block diagram of an algorithm that computes the true mean of a set of points on the fly*

```
double mean(reset, data)
int   reset;
double data;
{
  static double xhat, ki;

  return reset ? (ki = xhat = 0)
         : (xhat += (data \sc0\ xhat) / ++ki)
         ;
}
```

Example 9: *A C function that implements the algorithm shown if Figure 1*

Availability

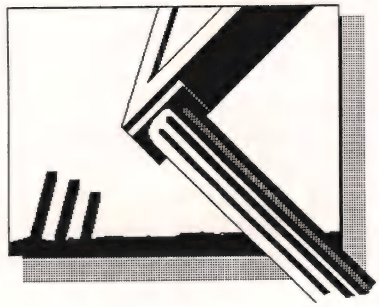
All the source code for the multitasking kernel described both this and next month (including the priority-queue stuff) is available for $30 on an IBM-PC 5¼″ disk from Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705. Include local sales tax if you're ordering from California. In addition to the kernel code and the priority queue routines, the disk includes an enhanced version of the curses window I/O package described in the July 1987 C Chest. Because the enhanced curses uses direct video reads and writes rather than going through DOS, it's useful in multitasking applications that can't use the DOS I/O functions. This version of curses supports overlapping windows (though you can only write to the top one) and lets you delete and move windows. In addition, it lets you create boxed windows (Unix's curses doesn't).

DDJ

(Listings begin on page 110.)

THE FORTH COLUMN

New Forth Sources, a Bibliography, and String Extensions for Forth–83



In the news this month is FORCE, Harris Semiconductor's version of the Novix NC4016 Forth processor. FORCE stands for Forth-optimized RISC computing engine. Chuck Moore's original hardware design has been copied into the Harris cell library, where it can be rapidly moved to various semiconductor technologies. Harris has fixed the NC4016 bugs and has added some extra features, such as byte addressability. FORCE is intended to be the smart heart of custom VLSI chips for knotty problems with high-speed solutions.

According to Dave Williams of Harris, a FORCE-based, real-time control processor (RTCP) will be available in the first quarter of 1988. The RTCP chip will contain an interrupt processor, 256-word data and return stacks, and a 16 × 16 hardware multiply. (Rumor has it that it will be offered on an IBM PC card shortly thereafter.) Dave says the chip will run at 15 MHz, or more than 15 million instructions per second. FORTH Inc. intends to support the chip with a variation of the same polyFORTH it developed for the Novix NC4016. This Forth includes an optimizing compiler that can pack sequential instructions together so that they run simultaneously. It also includes extensive fixed, fractional, and floating-point math support and a flexible nonlinear curve fitter (the mathematical equivalent of a monkey wrench).

Speaking of Forth chips, Dr. C. H.

by Martin Tracy

Ting's *More on NC4000, Volume 5,* is now available. This newsletter contains 80 pages of technical nitty gritty on the Novix NC4016 (originally numbered the NC4000). Volume 5 contains a reprint of "The FORCE Toolbox" by Dave Williams of Harris Semiconductor. You can order *More on NC4000* for $15 from Offete Enterprises Inc. at (415) 574-8250. Back issues are still available.

Also available from Dr. Ting at Offete is the *F83 Reference Manual* (1987). This is strictly a reference manual and is meant to accompany the popular public-domain Laxen/Perry F83 Forth. Words are arranged by topic, and each word has a one-line description. A total of 400 words are described in 40 pages, which include an index and a complete catalog of Offete's other publications—all this for only $10.

What's surprising is not that 400 words are included but that more than 600 words are left out! (The Forth-83 Standard only requires about 200 words.) To quote Dr. Ting: "The only problem with F83, like any good looking and hard working wife, is that it is too wordy."

GENie Forth Forum

The GENie (General Electric Network for Information Exchange) Forth Forum is now in operation. This new Forth-oriented bulletin board is sponsored by the Forth Interest Group (FIG). GENie is reportedly the largest electronic information exchange network, with local-access phone numbers from most major cities.

Alan Furman writes: "Use 1,200 bps, even parity, 7 bits, 1 stop bit, half-duplex (echo on). Dial up the sign-on modem line: (800) 638-8369. It helps to record the session on disk as there is a lot of scrolling off. After *CONNECT,* type *HHH* (without CR) or just wait 5–10 seconds. The *U#=* prompt will come on. At the prompt, type *XJM11849,GENIE* (and a CR). When the menu comes on, get the local node and billing information before signing up."

Signing up with this number results in the FIG member's discount deal: first three hours free (but regular $18 sign-up fee). The easiest way to sign up is by credit card number. GENie will make a confirming phone call a few days later. When they do, have a 3- to 12- character moniker ready (no embedded spaces). You can change your name later, but it will cost you $10. The $18 initial fee includes a nicely packaged users' manual that will arrive in about a week.

Once you are on GENie, type *FORTH* at the command line to go directly to the Forth conference. The sysops are Dennis Rufer (D.RUFFER), Scott Squires (S.W.SQUIRES), and Gary Smith (GARY-S). Use the *ATT* command to see who else has "attended." The categories are:

1. FIG Bulletin Board
2. FIG Real Time Conference
3. FIG Software Library
4. About the Roundtable
5. Roundtable News

You can download a copy of Laxen/Perry F83 Forth or one of the many files contributed by Gerald Shifrin of the East Coast Forth Board. Some categories are open to FIG members only.

GENie is definitely a non-prime-time activity at $5/hour; prime time costs $35/hour. User assistance is available until 9:00 PM Pacific Time at (800) 638-9636.

Forth Bibliography

The third edition of *A Bibliography of Forth References* (1987) is now available from the Institute for Applied Forth Research, P.O. Box 27686, Rochester, NY 14627. This 2,000-entry bibliography references arti-

THE FORTH COLUMN
(continued from page 144)

cles and books from Forth's dim origins up to January 1987. Articles are indexed both by subject and author.

Because many Forth programmers love to write, the author index reads like a *Who's Who* of Forth. I use the subject index mostly, though. It's a great help in convincing software managers or potential customers that Forth has already been used successfully in their line of work.

Here is an example of using the index:

Q: What's a QUAN?

A: (433) The Quan Concept Expanded
(1120) Code Field Vectoring
(1538) High Speed, Low Memory Consumption Structures

Q: Has Forth been used for radar antenna programming?

A: (359) The Development of a Computerized Antenna Range Field
(879) Microprocessor Control of Automated Antenna Ranges
(1010) Ground Control Approach Radar Performance Monitoring

The bibliography was edited by Thea Martin and was generated entirely with Forth programs provided by Dick and Jill Miller of Miller Microcomputer Services (MMS). Thea writes:

"We used MMS' DATAHANDLER-PLUS and FORTHWRITE to produce this document. DATAHANDLER-PLUS is a powerful, flexible database written in Forth that interfaces to the MMS word processor, FORTHWRITE. All three editions of the Bibliography have been produced with these systems."

You can order a copy of the bibliography for $25 directly from the Institute, or you can get it from FIG ([408] 277-0668).

ANS Forth Meeting

The first meeting of ANSI X3J14 (ANS Forth) was held on August 3 and 4 at CBEMA headquarters in Washington, D.C. In attendance were:

Greg Bailey, Athena Programming (Novix Forth)
Gary Betts, Saba Technology
Ronald D. Braithwaite, The Tools Group
Richard Burton, National Bureau of Standards
Don Colburn, Creative Solutions Inc. (MacFORTH)
Chris Colburn, Creative Solutions Inc. (MacFORTH)
Ted Dickens, The Dickens Co.
John Dorband, NASA GSFC
Ray Duncan, Laboratory Microsystems Inc. (PC/FORTH, UR/FORTH)
Douglas Fishman, National Bureau of Standards
Lawrence P. Forsley, Laboratory for Laser Energetics (Rochester Conference, *JFAR*)
Charlie Keane, PPI
Guy M. Kelly (IEEE representative, chairman of the FST)
Charles H. Moore, Computer Cowboys (inventor of Forth)
Mike Nemeth, Computer Sciences Corp. (MD FIG, Goddard FUG)
David C. Petty, Digitel

James Rash, NASA GSFC
Elizabeth D. Rather, FORTH Inc. (polyFORTH)
Gerald A. Shifrin, MCI Telecommunications (ECFB sysop)
Bill Ragsdale, Dorado Systems (founder of FIG, figForth)
Robert Smith, Maxtor (FST secretary)
Martin Tracy, FORTH Inc. (MasterForth, *DDJ*)

The editorial comments are claims to fame and not necessarily current agenda. Ms. Rather served as the acting chair and Mr. Duncan as the acting secretary. Ms. Cathie Kachurik offered welcome assistance to the X3J14 Technical Committee (TC) on behalf of CBEMA.

The complete unofficial minutes of this meeting are available on the MCI ANS Forth bulletin board, with a copy on the East Coast Forth Board ([703] 442-8695). (Minutes become official when approved at the next meeting.) Unofficially, here are some of the highlights:

- Ms. Rather observed that a Forth standard should identify and document accepted practice and not be an instrument to advance the state of the art. She hoped that an ANS Forth would be a clear and complete statement of what Forth is and does, that it would require minimal changes to existing systems, that it would be universally accepted, and that it would lead to the acceptance of the Forth language as a professional instrument.
- The Forth-83 Standard, Chapters 1–12, without the appendices, was approved as the Basis Document. (The Basis Document is successively modified until it becomes the Draft Proposal.) All members of the TC will receive a Basis Document with all paragraphs numbered for their comments and review.
- A proposal to add floating-point math to the Scope of Work was defeated.
- Volunteers were solicited for officer positions. The ANSI SMC selects officers from this pool of volunteers. Ad hoc committees were created to research existing practice and identify major areas of noncompliance to Forth-83. (The Research Committee has since mailed a questionnaire to all identified producers of Forth.)

The next ANS Forth meeting is scheduled for November 11 and 12, just prior to the Forth Convention. By the time you read this, both will be over. To be a voting member on the ANS Forth TC, you must pay CBEMA a $200 fee and be prepared to attend four three-day meetings each year. You can follow the progress of the dpANS (draft proposal, pronounced "de-pants") on the MCI ANS Forth bulletin board, GENie, or through this column. For now, any technical proposals should be sent to the acting secretary (me) at FORTH Inc., 111 N. Sepulveda Blvd., Manhattan Beach, CA 90266.

Some Controlled Words

In the October 1987 issue of *DDJ*, I presented a Forth-83 software prelude for writing portable source code. You will need to add the words presented in that article to your Forth-83 Forth to enjoy the fruits of this column. Example 1, page 148, contains typical defini-

THE FORTH COLUMN
(continued from page 147)

tions, which you can adjust to fit your Forth.

Now we can define a (small) number of additional words that have come into general usage:

```
: \ >IN @ 64 + -64 AND >IN ! ;
                        IMMEDIATE
( "backslash" comment to the end
                         of line.)
: THRU ( n n2) 1+ SWAP DO I LOAD
                            LOOP ;
\ LOAD blocks n through n2.
```

These words are defined early on so that we can use them to compile other useful words. The "backslash" comment has two cousins:

```
: \\ 1024 >IN ! ; IMMEDIATE
\ comment to end of source block.
: \IF ( f ) 0= IF [COMPILE] \ THEN ;
\ conditional comment or interpret
                              line.
   IMMEDIATE
```

The first word, \\, stops interpreting or compiling a block immediately. Most Forths can use *EXIT* for this function. The second word, *\IF*, is both a convenient comment and a key for conditional execution. I'll discuss how you can use *\IF* in a moment.

In the meantime, consider another useful word:

```
: 2* ( n - n') DUP + ;
```

Many programmers assume this word is required by the Forth-83 Standard. (It isn't, but 2/ is.) By defining 2* as high-level Forth, we guarantee that we can use it in any Standard program. But chances are excellent that 2* is already in our dictionary as a *CODE* definition. If we redefine it, the new 2* will be much slower than the original. What we would really like to do is to define 2* only if it isn't already in the dictionary:

```
: NEED ( - f)
\true if the following word is already
\in the dictionary.
   32 ( ie blank) WORD FIND SWAP
                       DROP 0= ;
```

Now we can have our cake and eat it too:

```
NEED 2* \IF : 2* ( n - n') DUP + ;
```

NEED looks 2* up in the dictionary. If it isn't there, we define it in high-level.

There are a few caveats to this approach. *\IF* can only be used within a source block. *NEED* returns either of two truth values, 1 or –1, so be careful if you use it in logical expressions. Finally, we assume that if 2* is already in the dictionary, its function is to double a number and not to, say, print two stars (**) on the terminal.

Fortunately, the Forth-83 Standard has provided for this eventuality by including 2* in its Controlled Reference Words with the appropriate definition. Controlled Reference Words are not required by the Standard, but if they do appear, they must have the prescribed definition. Example 2, below, contains some other Controlled Reference Words.

Unfortunately, there are not very many words in the Controlled Reference Set. (The Standard also has Uncontrolled Reference Words, which are exactly that.) Of course, we can define or redefine any word we want to. This is one of Forth's

```
( Moves NEXT address to and from stack.)
: I> ( - a) COMPILE R> ; IMMEDIATE
: >I ( a )  COMPILE >R ; IMMEDIATE

( Even address alignment, if required.)
: ALIGN HERE 1 AND ALLOT ;
: REALIGN ( a - a') DUP 1 AND + ;

( Hides number of bytes per word.)
2 CONSTANT CELL
: CELL+ ( n - n') 2+ ;
: CELLS ( n - n') 2* ;

( Compiles self-reference.)
: RECURSE ( ...) ; IMMEDIATE

( Forces interpretation of the input stream.)
: INTERPRET ( ...) ;

( Discards return stack overhead of DO--LOOP.)
: UNDO  I> R> R> 2DROP >I ;
```

Example 1

```
NEED D2* \IF : D2* ( d - d') 2DUP D+ ;
NEED HEX \IF : HEX ( DECIMAL ) 16 BASE ! ;

NEED C, \IF : C, ( n ) HERE 1 ALLOT C! ;
NEED BL \IF 32 CONSTANT BL ( a blank)

NEED ERASE \IF : ERASE ( a n) 00 FILL ;
NEED BLANK \IF : BLANK ( a n) BL FILL ;

NEED .R \IF : .R ( n w) >R DUP 0< R> D.R ;
```

Example 2

```
: 2>R ( n n2)
\ pushes a pair on the return stack.
  COMPILE SWAP COMPILE >R COMPILE >R ;
 IMMEDIATE

: 2R> ( - n n2)
\ pops a pair from the return stack.
  COMPILE R> COMPILE R> COMPILE SWAP ;
 IMMEDIATE

: @EXECUTE ( ? ) @ EXECUTE ;

: AGAIN
\ used in a BEGIN-- AGAIN structure.
  0 [COMPILE] LITERAL [COMPILE] UNTIL ;
 IMMEDIATE

: DLITERAL  SWAP
  [COMPILE] LITERAL [COMPILE] LITERAL ;
 IMMEDIATE

: S>D ( n - d) DUP 0< ;
\ single to double number.

: WITHIN ( n min max - f)
\ true if min <= n < max.
  OVER - >R - R> U< ;

-1 CONSTANT TRUE
```

Example 3

greatest benefits—nothing is sacred (except Forth itself). Anytime we redefine a *CODE* definition, however, we lose speed needlessly.

The solution is to add a few selected high-performance words to the Controlled Reference Set. Because the Forth-83 Standard committee is not meeting at this time, we will have to do it ourselves. So, consider the words in Example 3, page 148, as belonging to the *DDJ* Forth Column Controlled Reference Word Set. These definitions should each be preceded by the appropriate *NEED* phrase. Any other definitions we require, we will simply define (or redefine) in high-level Forth. In other words, we won't be needing *NEED* anymore.

Strings

This month's topic of interest is implementing strings in Forth. Let's see how we can apply our new tools to this issue.

The most enthusiastic string package I have ever seen was written by George Hawkins and is available as the file FSTRINGS.ARC from the ECFB. George's package provides 66 definitions on 50 screens of source code, including a test suite. There are 21 pages of documentation with a glossary. The package is written in Forth-83 but uses the *BRANCH* experimental extension. If your Forth doesn't *BRANCH*, change the following definitions on screen 6:

```
: $, HERE OVER @ 2+ DUP ALLOT
                    ALIGN CMOVE ;
: ($LIT) I> DUP
  DUP CELL+ SWAP @ + REALIGN
                             >I ;
: [$LIT] COMPILE ($LIT) ,$ ; IMMEDI-
                               ATE
```

Once I made this change, it compiled the first time and added a modest 3.3K to my dictionary.

Although the FSTRINGS package is not a SNOBOL, it is suitable for medium-strength string processing, such as building concordance tables. We can approximate many of its functions using materials that lie readily at hand.

First, we need some way to make strings. In Forth-83, strings are stored in memory as counted strings. A counted string is referred to by a single address, which points to the count byte of the string. The count byte is followed by that many 1-byte ASCII characters. In some Forths, it may be padded with a blank or zero to the next even address. You can think of a counted string as a packed string that must be unpacked to a text string before it can be used. Text strings are referred to by two arguments: the address of the first character and the length of the string, with the length on top. Counted strings are converted to text strings with the *COUNT* operator. We will also need an operator to pack text strings into counted strings:

```
: PLACE ( a1 n a2) 2DUP ! 1+ SWAP
                           CMOVE ;
\ packs string a1 n into counted
                         string a2.
: ," 34 ( ie ASCII ") WORD COUNT
\ compiles following string at HERE
  DUP >R HERE PLACE R> 1+
                    ALLOT ALIGN ;
CREATE TEST ," This is a test."
```

THE FORTH COLUMN
(continued from page 149)

```
TEST COUNT TYPE This is a test. ok
```

We can compile strings as literals into colon definitions in this way:

```
: (") I> COUNT 2DUP + REALIGN
                              >I ;
: " COMPILE (") ," ; IMMEDIATE
\ fundamental string compiler.
: TEST " This is a literal." TYPE ;
TEST This is a literal. ok
```

I have included *COUNT* in the run-time action of a string literal. Several Forths include the *"* operator but not all of them *COUNT* the string. Nonetheless, because we are redefining it ourselves, it will behave as we expect.

We can also make a string by *EXPECT*ing it from the user into *PAD*:

```
( For example:)
: TEXT ( – a n)
   PAD 80 EXPECT PAD SPAN @ ;
: NAME?
     CR ." Sign in please: " TEXT
     CR ." Your name is : " TYPE ;
```

Substrings are trivial in Forth:

```
: /STRING ( a n n2 – a' n')
\ shorten string a n by n2 characters.
   ROT OVER + ROT ROT – ;
: TEST " Catatonic" ;
TEST 6 – TYPE Cat ok
TEST 4 /STRING TYPE tonic ok
TEST 5 /STRING 2– TYPE on ok
```

We can make the *"* literal *STATE*-smart to make testing easier. While we're at it, let's have a *STATE*-smart ASCII, too:

```
: ASCII ( – c) BL WORD 1+ C@
\ value of following character.
   STATE @ IF [COMPILE] LITERAL
                          THEN ;
  IMMEDIATE
: (") I> COUNT 2DUP + >I ;
: " ( – a n) STATE @
   IF COMPILE (") ,"
   ELSE ASCII " WORD COUNT >R
      PAD I CMOVE PAD R> THEN ;
IMMEDIATE
ASCII A EMIT A ok
" Simplicity" TYPE Simplicity ok
```

Numbers are easily made into strings with the flexible "sharp" operators: <#, #, #s, *HOLD*, *SIGN*, and #>. Making strings into numbers, however, is a little harder. We'll use a syntax suggested by Stephen Pelc ("Proposed Standard Changes," *FORML Conference*, 1986):

```
VARIABLE DPL
: VAL? ( a n – d 2 , n 1 , 0)
\ string to number conversion.
   PAD OVER – SWAP OVER >R
                         CMOVE
BL PAD C! PAD DPL ! 0 0 R>
DUP C@ ASCII – = DUP >R – 1–
BEGIN CONVERT DUP C@ DUP
                      ASCII : =
   SWAP ASCII , ASCII / 1+ WITHIN
                             OR
WHILE DUP DPL ! REPEAT
R> SWAP >R IF DNEGATE THEN
PAD 1– DPL @ – DPL ! R> PAD =
IF DPL @ 0> IF DROP 1 ELSE 2
                          THEN
ELSE 2DROP 0 THEN ;
```

When the smoke clears, there will be a 2 on the stack if the number is a double, a 1 if it is single, and a 0 if it is invalid. The number, if any, will be on the stack under the flag. A number containing punctuation from the set +,–./: is a double number, with *DPL* set to the number of places to the right of the rightmost punctuation. A number without punctuation is a single number, and *DPL* is set to a negative value.

```
" 123,456" VAL? . 2 ok
D. 123456 ok
DPL @ . 3 ok
```

Many Forths already support automatic single- and double-number conversion, but there is no general agreement on the syntax. The Forth-83 *DPL* is one of the Uncontrolled Reference Words, which means we can't count on it. Hopefully, this issue will be taken up by the ANS Forth technical committee.

Because strings are often converted to double numbers, irrespective of punctuation, we could use a simpler conversion primitive based on *VAL?*:

```
: VAL ( a n – d f) VAL? 3 < AND DUP
\ string to double number.
\ True if number is valid.
   IF 1 = IF S>D THEN TRUE EXIT
                           THEN
```

THE FORTH COLUMN
(continued from page 150)

0 0 ;

SKIP and *SCAN* are two important string search primitives that appear in several Forths as factors of the word *WORD*:

SKIP (a n c – a' n')
\ shortens a string to the first position
\ unequal to c.
SCAN (a n c – a' n')
\ shortens a string to the first position
\ equal to c.

SKIP and *SCAN* mimic the 8086 *SCAS* instruction. Their high-level definitions are given in the accompanying source screens (see Listing One, page 124). *SCAN* is especially useful for lexing and parsing strings:

```
( For example:)
: LEX ( a n c – a2 n2 a3 n3)
\ splits string at c, right most string
\ on top. Either string can have 0 length.
   >R 2DUP R> SCAN ROT OVER –
ROT ROT DUP 0> NEGATE /STRING ;
" FORTH.COM" ASCII . LEX
2SWAP TYPE SPACE TYPE FORTH COM ok
```

CTO"" ("C-to-quote") is an often neglected primitive for converting a character to a string:

```
: CTO"" ( c – a 1) CTEMP C! CTEMP 1 ;
ASCII A CTO"" TYPE A ok
```

```
-TEXT ( a n a2 - -1 , 0 , 1)
\ -1 if string a n < a2 n , 0 if equal,
\ and 1 if >.

COMPARE ( a n a2 n2 - -1 , 0 , 1)
\ -1 if string a n < a2 n2 , 0 if equal,
\ and 1 if >.

-MATCH ( a n a2 n2 - offset 0 , ? -1)
\ position of string a2 n2 in a n.
\ Offset is 0 if a n is found in 1st
\ position. True with invalid offset
\ if a2 n2 isn't in a n.

: ANIMAL " ANIMAL" ;

" ANIMATE" ANIMAL COMPARE . 1 ok

" ANT" ANIMAL COMPARE . -1 ok
ANIMAL " IMA" -MATCH . . 0 2 ok
ANIMAL " XYZ" -MATCH . . -1 ????? ok
```

Example 4

The remaining string operators (shown in Example 4, left) are used to compare strings and to find the occurrence of one string in another. Their analogs are often found in line and screen editors.

The final string operator, *EVAL*, is also the most powerful. *EVAL* interprets a string. The following definition of *EVAL* should work on your Forth-83 system. I say "should" because there is some question as to whether the Standard words *>IN*, *BLK*, *TIB*, and *#TIB* control interpretation or simply describe it. You will need to test this word on your system.

```
: EVAL ( a n) \ interprets the string.
  DUP >R TIB SWAP CMOVE R@ #TIB !
  0 >IN ! 0 BLK ! INTERPRET R> >IN ! ;
: TEST " LATER" EVAL ;
: LATER ." A forward reference." ;
TEST A forward reference. ok
```

This completes the string package, and you will find a complete listing of it in Listing One. It compiles to about 1K of dictionary space.

Availability

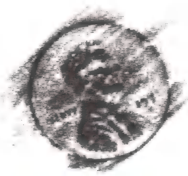
Most of the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listing begins on page 124.)

Vote for your favorite feature/article.
Circle Reader Service **No. 7.**

Remember,



Smart Buying Decisions

Start with DDJ

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #200, DALTON, MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Reader Service Dept.
P.O. Box 507
Dalton, Mass. 01227

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

Name ______________________________
Title ______________________________
Company ____________________ Phone (___) __________
Address ______________________________
City/State/Zip ______________________________

1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

December '87: Use before March 31, 1988

1. Did you buy this issue on a newsstand? ( ) Yes ( ) No
2. Are you a subscriber? ( ) Yes ( ) No
3. Have you purchased a product as a result of seeing it advertised in *Dr. Dobb's Journal?* ( ) Yes ( ) No

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

Name ______________________________
Title ______________________________
Company ____________________ Phone (___) __________
Address ______________________________
City/State/Zip ______________________________

1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

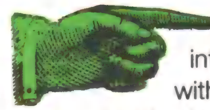
December '87: Use before March 31, 1988

1. Did you buy this issue on a newsstand? ( ) Yes ( ) No
2. Are you a subscriber? ( ) Yes ( ) No
3. Have you purchased a product as a result of seeing it advertised in *Dr. Dobb's Journal?* ( ) Yes ( ) No

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

For Free Info ...

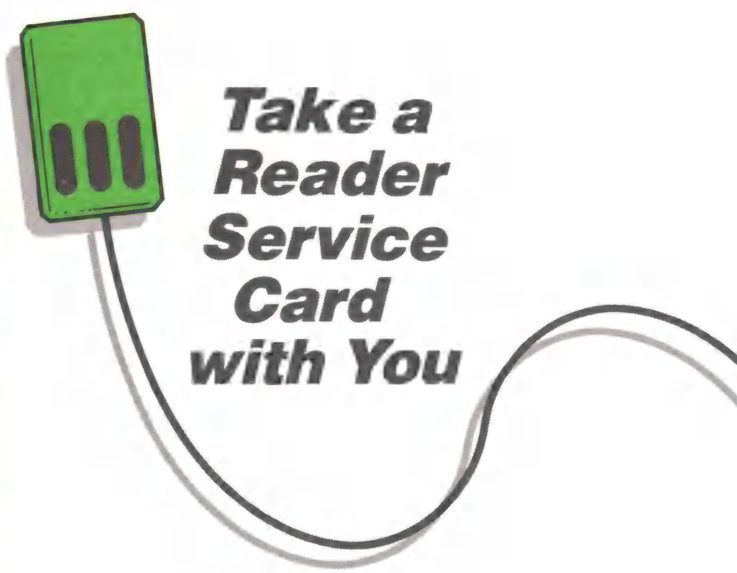
Start Here



Smart buyers start with *DDJ*'s free information card, a shopping center filled with information about the products and services advertised in this very issue: everything from software and systems to peripherals and professional support services.

And smart buyers can use this free information card to quickly and easily gather a comprehensive file of facts, figures and product specs to sort out competing claims. Using *DDJ*'s free information card can prevent you from making the wrong, costly buying decision.

Be a smart shopper. Complete and mail this postage paid card today!



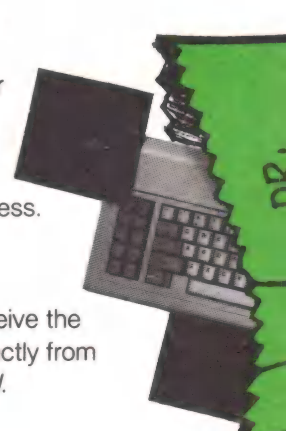
Take a Reader Service Card with You

It's Easy as ...

1. Circle the appropriate free information numbers, referring to the advertiser index for more information.

2. Fill in your name and address.

3. Mail today—postage is absolutely free. You'll receive the product information you need directly from the manufacturers, thanks to *DDJ*.



The Advertiser Index

Advertiser Name	Page #	RS#
Addison Wesley	119	195
AI Architects	112	187
Aker Corporation	44	124
Aldebaran Laboratories	47	126
Apex Software Corporation	109	185
ASI/American Software International	79	150
Aspen Scientific	37	120
AT&T Electronic Photography & Imaging	10-11	106
Austin Code Works	143	206
Azatar Computer Systems, Inc.	149	211
Blaise Computing	2	102
Block Island Tech.	147	210
Borland International	1	101
Bryte Computer	83	155
Burton Systems Software	90	163
C Users Group	90	162
Centurion Discount Software	156	215
CNS, Inc.	92	167
Coder's Source (The)	82	154
Coder's Source (The)	56	133
Cogent Software Ltd	79	151
Compaq Computer Corporation	32A	*
Comptech	42	122
Compu View	25	111
Computer Technology Group	94	169
Creative Programming	93	*
Crosstalk Communications	C6	219
DDJ Subscriptions	136	*
Desktop A.I.	94	170
Digitalk	70-71	144
Disk Software	115	191
Ecosoft, Inc.	133	202
Essential Software	C5	218
Fair-Com	41	121
GenSoft Systems	81	153
Gimpel Software	130	*
Golden Bow Systems	103	180
Greenleaf Software	27	112
Guidelines Software	45	125
Hauppauge Computer Works	60	136
IMSI	57	134
Intel Corporation	58-59	135
JYACC	53	132
Kadac Products Ltd.	117	193
Komputerwerk	113	189
Laboratory Microsystems, Inc.	73	146
Lattice, Inc.	69	143
Logic Process Corporation	115	192
Logitech, Inc.	9	105
Lugaru Software Ltd.	152	213
M Street Software	50	129
M&T Catalog of Books & Software Tools	104A	*
Machine Independent Software Corp.	105	181
Magma Systems	91	165
Manx Software Systems	7	104
Meridian Software Systems	51	130
Metagraphics Software Corporation	101	179
MetaWare Incorporated	85	158
Micro Way	75	148
Microprocessors Unlimited	84	157
Microrim	128-129	199
Microsoft	135	203
MMC AD Systems	137	204
Mortice Kern Systems, Inc.	32	115
MSJ Subscriptions	40	*
Nanosoft Associates	74	147
Norton Utilities (The)	150	212
Norton Utilities (The)	4-5	103
Nostradamus	107	182
NWP Intellegent Solutions, Inc.	33	116
Oakland Group, Inc. (The)	145	207
Oasys	111	186
Oregon Software	83	156
Peacock Systems	99	178
Periscope Co. Inc.	28	113
Pharlap	99	176
PIM Publications	117	194
Polytron	15	108
Production Language Corp.	92	166
Programmer's Connection	157-159	217
Programmer's Paradise	66	141
Programmer's Power Pack	76	*
Programmer's Shop (The)	97	172-174
Programmer's Shop (The)	96	171
QCAD Systems, Inc.	50	128
Quantum Software	72	145
Quarterdeck Office Supplies	13	107
Raima Corporation	127	*
Roundhill Computer Systems LTD	77	149
SAS Institute	89	*
Santa Rita Software	131	201
Scantel Systems Limited	63	139
Scientific Endeavors	87	160
Secom Information Products Co.	99	177
Seidl Computer Engineering	61	137
Semaphore, Inc.	113	190
Semi-Disk Systems	139	205
Sharpe Systems Corporation	155	214
SLR Systems	52	131
Softfocus	146	208
SoftScience Corporation	43	123
Software Connections Inc.	80	152
Software Garden Inc.	67	142
Software Link	21-23	110
Software Security, Inc.	36	119
Solution Systems	29	114
Solution Systems	49	127
Stony Brook Software, Inc.	121	196
Tangent Designs, Inc.	113	188
Tenon Software	63	*
Texas Instruments	16-17	*
Tool Makers (The)	156	216
TSF (The Software Family)	35	118
Turbo Power Software	34	117
Turbo Tech Report	95	*
Wallsoft	125	197
Whitewater Group (The)	65	140
Wordcraft	87	159
Wyte Corporation	147	209
Xenosoft	90	164

*The advertiser prefers to be contacted by phone; consult ad.

Advertising Sales Offices

Midwest
Charles Shively (415) 366-3600

Northern California/Northwest
Lisa Boudreau (415) 366-3600

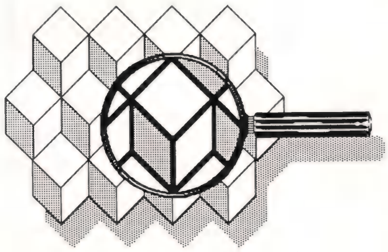
Northeast
Cynthia Zuck (718) 499-9333
Martha Brandt (415) 366-3600

Southern California/AZ/NM/TX
Michael Wiener (415) 366-3600

Director of Marketing and Advertising
Ferris Ferdon (415) 366-3600

Telemarketing Representative/SE/SW USA
Cheri Blum (714) 761-0294

OF INTEREST



Operating Systems

VenturCom has announced Venix System V 2.3, a real-time Unix operating system for the IBM PC/AT, HP Vectra Plus, and Compaq Deskpro 386. Enhancements include impoved memory management to minimize I/O; a more efficiently coded kernel; a larger, 1,024-byte file system; and an extended buffer cache.

Venix 2.3 also features real-time extensions, adherence to AT&T standards, EGA support, AT&T binary compatibility, developers' tools that include a large-model C compiler, and Ethernet TCP/IP support.

The full system is priced at $990 for single-quantity purchases but is being offered at an introductory price of $600 for a two-user system until December 31, 1987. It comes with an indexed, four-volume set of documentation and 60 days of toll-free telephone support. Reader Service No. 16.
VenturCom Inc.
215 First St.
Cambridge, MA 02142
(617) 661-1230

PC-MOS/386, Version 1.02, from **The Software Link**, now supports IBM PCs; IBM PC/ATs; and compatible 8088-, 8086-, and 80286-based systems. PC-MOS/386 offers file sharing, user security, print spooling, disk caching, NETBIOS emulation, EMS emulation, interrupt-driven serial I/O, enhanced command-line recall and editing, support for very large disk volumes, an enhanced directory structure, sophisticated command processing, a full-screen text editor, and on-line help. Features that are specific to the 80386 CPU, such as support for 32-bit native-mode applications, are now isolated in a single system driver file.

PC-MOS/386 is available on both $5^1/_4$- and 3.5-inch disks. Prices range from $195 for a 1-user version to $995 for a 25-user version. Registered users of earlier versions can upgrade to Version 1.02 at no charge. Reader Service No. 17.
The Software Link
3577 Parkway Ln.
Atlanta, GA 30092
(404) 448-5465

Digital Research is now shipping enhanced versions of Concurrent DOS 386 and Concurrent DOS XM (Expanded Memory).

Concurrent DOS 386 supports most PC-DOS/MS-DOS applications and more than 700 business applications written specifically for Concurrent DOS. Windowing capabilities are provided, allowing up to four applications to run from the primary console. It is available in a three-user system configuration, priced at $395, or in a ten-user version for $495.

Concurrent DOS XM offers improved EGA support, enhanced support for the IBM PC/AT keyboard, disk formatting for up to four partitions per hard disk, and support for the AST-Four Port/DOS card and the four-port Hostess Multiport Network Adapter. The three-user system is available for $295 and the six-user version for $395. Reader Service No. 18.
Digital Research
P.O. Box DRI
Monterey, CA 93942
(408) 649-3896

The Hyperspace Z-System for HD64180-compatible and Z280-compatible microprocessors is now available from **Echelon**. The Hyperspace Z-System is a CP/M 2.2-compatible operating system that includes ZCPR 3.3, ZRDOS 2.0, and a sample BIOS for 64180 machines. The system features a large free memory area of 57.25K. The retail price is $195. Reader Service No. 19.
Echelon Inc.
885 N. San Antonio Rd.
Los Altos, CA 94022
(415) 948-3820

UniPress Software has launched the Unix Training Center, which offers 22 courses to help users and organizations learn to use Unix. The center personnel can provide needs analyses, curriculum design, original course development, and instructor training. The Unix Training Center's classes can be given either at an organization's facility or at an off-site meeting area determined by a client. Classes are offered on-site for a fee of $100 per day per student, with a minimum fee of $1,300 per day. Reader Service No. 20.
UniPress Software
2025 Lincoln Hwy.
Edison, NJ 08817
(201) 985-8000

The Wendin-DOS Application Developer's Kit is now available from **Wendin**. The kit allows application programmers to develop multitasking applications based on VAX/VMS system services. The new QIO (Queued I/O) and RMS (Record Management System) services replace antiquated MS-DOS calls and induce more programming flexibility. Other serices support shared memory for interprocess communication, memory-resident pipes, semaphones, file locking, file permissions, extended memory access, swapping, and control of multiple terminals. The kit sells for $99. Reader Service No. 21.
Wendin Inc.
P.O. Box 3888
Spokane, WA 99220-3888
(509) 624-8088

Microprocessor Engineering has released its OS-9 implementation of the MPE/Nautilus Cross Compiler. MPE/Nautilus features interactive debugging that allows development of the target system in RAM before committing to ROM. Written in Forth, MPE/Nautilus includes automatic handling of defining words and support for forward referencing. Reader Service No. 22.
Microprocessor Engineering Ltd.

(continued from page 154)

133/133a Hill Ln.
Shirley, Southampton
England S01 5AF
0703-631441

Hardware

Lloyd I/O is now shipping its OMEGA MC68020-based workstation. The workstation provides integral floating-point math support via the MC68881 math coprocessor. It also includes 1 megabyte of zero-wait-state; nonvolatile static RAM; the OS-9, 68K, real-time, multitasking operating system; and a C compiler. The standard system configuration supports up to four users; further users are supported by optional I/O expansion boards. The base price for the workstation is $4,750. Special system configurations, higher clock speeds, and hardware options are available for an additional cost. Reader Service No. 23.

Lloyd I/O Inc.
19535 Northeast Glisan St.
P.O. Box 30945
Portland, OR 97230
(800) 227-3719

An 80386-based multiuser system running the Xenix System V operating system, the Altos 386 Series 2000, is available from **Altos Computer Systems**. The Series 2000's design features modular architecture that allows memory and storage to be easily expanded. It also includes an intelligent file processor subsystem; system memory; a communications processor; an ESDI hard-disk drive; a 1.6 megabyte, $5^{1}/_{4}$-inch floppy-disk drive; a 60-megabyte streaming magnetic tape unit; and an Altos V terminal. The 80386 processor runs at 16 MHz with a 32K data and instruction cache, an intelligent file processor, and an 80387 floating-point coprocessor.

The Altos 386 Series 2000 is available in four configurations ranging in price from $25,000 to $30,000. Reader Service No. 24.

Altos Computer Systems
2641 Orchard Pkwy.
San Jose, CA 95134
(408) 946-6700

DDJ

Upgrade your technology

The software technology available to programmers of IBM-compatible personal computers is truly amazing. And newer, more powerful development packages appear all the time. But until now, finding out about these important products has been a difficult and time consuming task.

FREE Buyer's Guide. The New Programmer's Connection Buyers Guide contains individual descriptions of over 500 titles of programmer's development software by over 150 manufacturers. Each description covers major product features as well as any software or hardware requirements and version numbers. In the box on the right are some examples of the types of descriptions you'll find in our Buyer's Guide.

No Hidden Charges. The low discount prices in our Buyer's Guide are all you pay. We don't charge extra for domestic UPS Ground shipping, credit cards, COD orders, purchase orders, sales tax (except Ohio) or special handling (except for non-Canadian international orders).

Guarantees. We offer FREE 30-day no-risk return guarantees and 30-day evaluation periods on most of our products.

Latest Versions. The products we carry are the latest versions and come with the same manufacturer's technical support as if buying direct.

Large Inventory. We have one of the largest inventories of programmer's development products in the industry. Most orders are shipped within 24 hours.

Noncommissioned Staff. Our courteous salespeople are always ready to help you. And if you aren't sure about your needs, our knowledgeable technical people can give you sound, objective advice.

Experience. We've specialized in development software for IBM-compatible personal computers since 1984 and are experienced at providing a full range of quality products and customer services.

How to Get Your Copy. There are three ways for you to receive your FREE copy of the Programmer's Connection Buyer's Guide: 1) Use the reader service card provided by this journal; 2) Mail us a card or letter with your name and address; or 3) Call one of our convenient toll free telephone numbers.

If you haven't yet received your Programmer's Connection Buyer's Guide, act now. Upgrading your programming technology could be one of the wisest and most profitable decisions you'll ever make.

Announcing Our Super Year-end SALE on selected Microsoft Products

until December 31, 1987

New Version 5.0 Microsoft C Optimizing Compiler 5.0

List $450 Ours $269

Sale Price $259*

The Microsoft C Optimizing Compiler is a high performance, industry- standard C language development system. It uses many state-of-the-art optimizing techniques including: in-line code generation; loop invariant expression removal; automatic register allocation of variables; elimination of common subexpressions; improved constant folding & value propagation; and efficient coding techniques covered in the documentation. This extensive package also includes the new Microsoft QuickC Compiler (see separate description) that lets you quickly develop programs with its integrated environment and extremely fast compilation speeds. Microsoft C features: library routines that implement UNIX System V and most of the proposed ANSI standard libraries; extremely fast and optimized code generation; five memory models including HUGE; mixed models using NEAR, FAR and HUGE pointers; huge programs up to one MB and huge arrays larger than 64K; and source and object compatibility with XENIX. The package also includes Microsoft CodeView, an advanced source level debugger that features: support for overlays created with Microsoft Link; Expanded Memory Specification (EMS) support; access to source level and symbolic debug information from your Microsoft C, FORTRAN and MASM programs; simultaneous view of source code and assembly; display of local and global variables and expressions; animation and single-step; breakpoints and watchpoints; code and output screens; display of CPU registers and flags; and support for keyboard or optional mouse.

Requires 384K memory. Hard disk recommended. Version 5.0.

New Version 5.0 Microsoft Macro Assembler

List $150 Ours $93

Sale Price $89*

The Microsoft Macro Assembler (MASM) is the fastest and most complete assembler development system ever created. Written in Microsoft C, MASM assembles programs at blinding speeds. Make your programs run faster by linking assembly language subroutines to your Microsoft QuickBASIC, C, QuickC, FORTRAN or Pascal programs. A broad range of examples in the completely revised and expanded documentation makes it easy to write assembly language routines, plus template programs are online so all you have to do is plug in your code and go – MASM takes care of the interfacing. There is also a complete reference to the instruction set containing extensive examples. The Macro Assembler itself has sped up by 25-40% and the linker has doubled its speed since version 4.0. Now there is support for the 80386 and 80387 instruction set. Microsoft CodeView, the window-oriented, source-level debugger gives you total control over debugging. This highly acclaimed debugger is ideal for debugging your assembly language subroutines called from QuickBASIC, C, QuickC, Pascal or FORTRAN. Debug using your original source code and comments. Watch the value of variables and expressions change while your program executes and view registers and flags.

Requires 256K memory. Version 5.0.

*Plus a Special Rebate from Microsoft

Purchase any two of the following products before January 31, 1988 and receive an additional $30 rebate directly from Microsoft. Or purchase any three and receive a $50 rebate:

- Microsoft COptimizing Compiler
- Microsoft Macro Assembler
- Microsoft QuickC
- Microsoft QuickBASIC

CALL TOLL FREE

USA:	800-336-1166
Canada:	800-225-1166
Ohio & Alaska (Collect): ..	216-494-3781
International:	216-494-3781
Telex:	9102406879
Easylink:	62806530

Programmer's Connection
7249 Whipple Avenue NW
North Canton, OH 44720

Please turn the page for our latest price list and ordering information. →

PROGRAMMER'S Service CONNECTION
QUALITY INTEGRITY SUPPORT PRICE

The Free Programmer's Connection Buyer's Guide.

CIRCLE 217 ON READER SERVICE CARD

ai - expert systems

	List	Ours
1st-CLASS *by Programs in Motion*	495	399
EXSYS *Development Software by EXSYS*	395	289
EXSYS Runtime System	600	469
LEVEL5 *by Information Builders*	685	569
Logic-Line Series *All varieties by Thunderstone*	CALL	CALL
VP EXPERT *by Paperback Software* New	100	63

ai - lisp language

	List	Ours
muLISP-87 Interpreter *by Soft Warehouse* New	300	199
muLISP-87 Interpreter & Compiler New	400	269
Q'Nial *Various by NIAL Systems*	CALL	CALL
Star Sapphire LISP *Compiler by Sapiens*	495	429
TransLISP PLUS *from Solution Systems*	195	125

ai - prolog language

	List	Ours
Arity Combination Package	1095	979
Expert System Development Pkg	295	229
File Interchange Toolkit	50	44
PROLOG Compiler & Interpreter	650	569
Screen Design Toolkit	50	44
SQL Development Package	295	229
Arity PROLOG Interpreter	295	229
Arity Standard Prolog	95	77
LPA microPROLOG *All Varieties*	CALL	CALL
MPROLOG Language Primer *LOGICWARE*	50	45
MPROLOG P500 *by LOGICWARE*	495	395
MPROLOG P550 *w/Primer by LOGICWARE*	220	175
Turbo PROLOG *by Borland Intl*	100	64
Turbo PROLOG Toolbox *by Borland Intl*	100	64

ai - smalltalk language

	List	Ours
Smalltalk/V	100	84
EGA/VGA Color Option	50	45
Goodies Diskette	50	45
Smalltalk/Comm	50	45

ai - texas instruments

	List	Ours
Arborist *Decision Tree Software*	595	519
PC Scheme *Lisp* New Version	95	77
Personal Consultant Easy New Version	495	435
Personal Consultant Image	495	435
Personal Consultant Online	995	869
Personal Consultant Plus New Version	2950	2589
Personal Consultant Runtime	95	84

ada language

	List	Ours
AdaVantage *GSA-Validated by Meridian Software*	795	735
AdaVantage Utility Packages	50	47
DOS Environment Package	50	47
Janus/ADA C Pak *by R&R Software*	95	84
Janus/ADA D Pak *by R&R Software*	1250	1059
Janus/ADA ED Pak *by R&R Software*	395	349

apl language

	List	Ours
APL*PLUS PC *by STSC*	695	495
APL*PLUS PC Spreadsheet Mgr *by STSC*	195	139
APL*PLUS PC Tools *by STSC*	CALL	CALL
APL*PLUS PS/2 *by STSC*	695	495
ATLAS*GRAPHICS *by STSC*	450	329
Financial/Statistical Library *by STSC*	275	189
Pocket APL *by STSC*	95	69
STATGRAPHICS *by STSC*	895	649

assembly language

	List	Ours
386 ASM/LINK *Cross Asm by Phar Lap*	495	389
8088 Assembler *w/Z-80 Translator by 2500 AD*	100	89
ASMLIB *Function Library by BCSoft*	149	125
asmTREE *B-Tree Dev System by BCSoft*	395	329
Cross Assemblers *Various by 2500 AD*	CALL	CALL
EZASM *by C Source*	70	59
Microsoft Macro Assembler SALE	150	89
Turbo Debugger *by Speedware*	89	79
Turbo Editasm *by Speedware*	99	84
Visible Computer: 8088 *by Software Masters*	80	64

basic language

	List	Ours
db/Lib *for QuickBASIC by AJS Publishing*	139	119
Finally *by Komputerwerk*	99	85
MACH 2 *by Micro Help*	69	55
Microsoft QuickBASIC 4.0 SALE	99	59
QBase *Relational Database by Crescent*	89	79
Quick-Tools *by BCSoft*	130	109
QuickPak *by Crescent Software*	69	59
Scientific Subroutine Library *by Wiley*	125	99
Screen Sculptor *by Software Bottling*	125	91
Stay-Res *by MicroHelp*	69	55
True BASIC	100	79
True BASIC *w/Run-time* Special Price	200	99
True BASIC 3D Graphics	50	41
True BASIC Developer's Toolkit	50	41
Turbo BASIC Compiler *by Borland Intl*	100	64

blaise products

	List	Ours
ASYNCH MANAGER *Specify C or Pascal*	175	135
C TOOLS PLUS/5.0	129	99
KeyPlayer *Super Batch Program*	50	45
LIGHT TOOLS *for Datalight C*	100	65
PASCAL TOOLS	125	95
PASCAL TOOLS 2	100	79
PASCAL TOOLS & TOOLS 2	175	135
RUNOFF *Text Formatter*	50	45
TURBO ASYNCH PLUS	100	79
TURBO C TOOLS	129	99
TURBO POWER TOOLS PLUS	100	79
VIEW MANAGER *Specify C or Pascal*	275	199

borland products

	List	Ours
EUREKA *Equation Solver*	167	105
Reflex: The Analyst	150	99
Sidekick	85	57
Superkey	100	64
Turbo Basic Compiler	100	64
Turbo Basic Database Toolbox	100	64
Turbo Basic Editor Toolbox	100	64
Turbo Basic Telecom Toolbox	100	64
Turbo C Compiler *(Call for support products)*	100	64
Turbo Lightning and Word Wizard	150	94
Turbo Lightning	100	64
Turbo Lightning Word Wizard	70	47
Turbo Pascal and Tutor New Version	CALL	CALL
Turbo Pascal New Version	100	64
Turbo Pascal Tutor	70	41
Turbo Pascal Database Toolbox	100	64
Turbo Pascal Developer's Toolkit	395	289
Turbo Pascal Editor Toolbox	100	64
Turbo Pascal Gameworks Toolbox	100	64
Turbo Pascal Graphix Toolbox	100	64
Turbo Pascal Numerical Methods Toolbox	100	64
Turbo Prolog Compiler	100	64
Turbo Prolog Toolbox	100	64

c language

	List	Ours
C-terp *by Gimpel, Specify compiler*	298	219
C Trainer *with Book by Catalytix*	122	87
DeSmet C *w/Debugger & Large case*	209	184
DeSmet C *w/Debugger only*	159	138
Eco-C88 Modeling Compiler *by Ecosoft* New	100	79
Instant C *by Rational Systems*	495	369
Lattice C Compiler *vers. 3.2 from Lattice*	500	265
Mark Williams Let's C *with FREE csd*	75	54
Microsoft C Compiler 5.0 w/CodeView SALE	450	259
Microsoft QuickC Compiler SALE	99	59
Optimum-C *by Datalight*	139	95
Turbo C Compiler *by Borland*	100	64
Uniware 68000/10/20 Cross Compiler *by SDS*	995	829

c utilities

	List	Ours
Blackstar C Library *by Sterling Castle*	125	98
C++ *by Guidelines w/version 1.1 kernel*	195	172
c-tree & r-tree Combo *by FairCom*	650	519
c-tree *ISAM File Manager*	395	315
r-tree *Report Generator*	295	239
Csharp Realtime Toolkit *by Systems Guild*	600	489
Curses Window Dev Pkg *by Aspen Scientific*	119	105
with Source Code	289	249
dBx *dBASE to C Translator by Desktop AI*	350	299
with Source Code	550	419
Flash-up *by Software Bottling*	90	78
GraphiC *Color version by Sci Endeavors*	350	274
HALO *Graphics by Media Cybernetics*	300	205
HALO Development Pkg *for Microsoft*	595	389
The HAMMER *by OES Systems*	195	129
PANEL/TC *for Turbo C by Roundhill*	129	95
PANEL Plus *by Roundhill*	495	395
PC Lint *by Gimpel Software*	139	99
RTC PLUS *Fortran to C by Cobalt Blue*	450	399
Sapiens V8 *Virtual Memory Manager*	300	265
Scientific Subroutine Library *by Wiley*	175	135
TE Text Editor *source by Sub Systems*	95	85
Vitamin C *by Creative Programming*	225	149
VC Screen *Forms Designer*	100	79
Zview *by Data Mgmt Consultants*	245	139

cobol language

	List	Ours
COBOLspII *by Flexus*	395	329
FPLIB *for Realia COBOL by BCSoft*	149	129
Micro Focus COBOL *See Micro Focus Section*		
Microsoft COBOL *See Microsoft Section*		
PCDT *by Pro-Code*	995	895
Realia COBOL with RealMENU	1145	899
Realia COBOL	995	783
RealCICS	995	783
RM/COBOL *by Ryan-McFarland*	950	639
RM/COBOL 85 *by Ryan-McFarland*	1250	895
RM/NET+5 *by Ryan-McFarland*	300	259
RM/Screens	395	334
SCREENIO *by Norcom*	400	379
screenplay *for COBOL by Flexus*	175	129

css products

	List	Ours
Combo Package *by Custom Software Systems*	199	175
PC/TOOLS *UNIX-like Utilities*	49	45
PC/VI *vi Editor*	149	99

database management

	List	Ours
Advanced DBMaster *by Macon Software* New	510	419
dBASE III Plus *by Ashton-Tate* New	695	389
dbIII Compiler *by WordTech Systems* New	CALL	CALL
dbSQL *by WordTech Systems* New	CALL	CALL
dbXL *by WordTech Systems* New	169	109
dFLOW *by Wallsoft* New	149	119
The Documenter *by Wallsoft* New	295	239
Genifer *by Bytel* New	395	279
Paradox 1.1 *by Ansa/Borland* New	495	359
Paradox 2.0 *by Ansa/Borland* New	725	525
Paradox Network Pack *by Ansa/Borland* New	995	725
Q&A *by Symantec* New	349	219
Quickcode PLUS *by Fox & Geller* New	295	179
Quickindex *by Fox & Geller* New	149	95
Quickreport *by Fox & Geller* New	295	179
QuickSilver *by WordTech Systems* New	599	349
R:Base 5000 *by Microrim* New	495	CALL
R:Base System V *by Microrim* New	700	CALL
\ rdb *by Robinson-Shafer-Wright* New	139	119
Tom Rettig's Library *by Tom Rettig & Assoc* New	100	79
UI Programmer *by Wallsoft* New	295	239
VP INFO *by Paperback Software* New	100	63

debuggers & profilers

	List	Ours
386 DEBUG *Cross Debugger by Phar Lap*	195	129
Advanced Trace-86 *by Morgan Computing*	175	115
Codesmith-86 *by Visual Age*	145	98
DSD87 *by Soft Advances*	125	79
MiniProbe *by Atron*	395	369
Periscope I *with Board by Periscope*	345	275
Periscope II *with NMI Breakout Switch*	175	139
Periscope II-X *Software only*	145	105
Periscope III *8 MHz version*	995	795
Periscope III *10 MHz version*	1095	875
The PROFILER *with Source Code by DWB*	125	89
TURBOsmith *Source debugger for Turbo Pascal*	99	89
The WATCHER *Profiler by Stony Brook*	60	51

disk utilities

	List	Ours
Back-It *by Gazelle Systems* New Version	130	115
Disk Optimizer *by Softlogic Systems*	60	55
Disk Technician *by Prime Solutions*	100	89
FASTBACK *by 5th Generation Systems*	179	129
Take Two Manager *United Software Security* New	139	119
Vcache *by Golden Bow Systems*	50	47
Vopt *by Golden Bow Systems*	50	47
Vfeature *by Golden Bow Systems*	80	74
Vfeature Deluxe *by Golden Bow Systems*	120	111
XenoCopy-PC *by XenoSoft*	80	69

dos utilities

	List	Ours
Advanced Norton Utilities	150	89
Command Plus *by ESP Software*	80	69
Desqview *from Quarterdeck*	130	115
FANSI-CONSOLE *by Hersey Micro*	75	62
Mace Utilities *Paul Mace Software*	99	89
MicroHelp Utility *by MicroHelp*	59	49
Norton Commander *by Peter Norton*	75	55
Norton Utilities *by Peter Norton*	100	59
OPAL *Shell Language by Software Factory*	99	89
Q-DOS II *by Gazelle Systems*	70	59
Taskview *by Sunny Hill Software*	80	55

essential products

	List	Ours
Breakout Debugger Only *Any language*	125	89
C Utility Library	185	119
Essential Communications	185	125
Essential Communications *with Break Out*	250	189
Essential Graphics	250	183
/*resident_C*/ New	99	85
with Source Code New	198	169
ScreenStar New	99	85
with Library Source Code New	198	169

forth language

	List	Ours
CFORTH *Native Code Compiler by LMI*	300	229
FORTH/83 Metacompiler *Specify Target*	750	599
PC/FORTH *by Laboratory Microsystems*	150	109
PC/FORTH+ *by Laboratory Microsystems*	250	199
Programmer's Package #1 *by LMI*	250	199
Programmer's Package #2 *by LMI*	350	279
Programmer's Package #3 *by LMI*	500	399
UR/FORTH *Also Available for OS/2 by LMI*	350	279
UR/FORTH Libraries	500	395

fortran language

	List	Ours
50 MORE: FORTRAN *by Peerless Scientific*	125	95
ACS Time Series *by Alpha Computer Service*	495	389
AUTOMATED PROGRAMMER *by KGK Automated*	995	949
Essential Graphics *by Essential Software*	250	183
Forlib-Plus *by Alpha Computer Service*	70	44
FORTRAN Addendum *by Impulse Engr*	95	85
FORTRAN Addenda *by Impulse Engr*	165	138
HALO *Graphics by Media Cybernetics*	300	205
I/O PRO w/No Limit Library *by MEF*	250	219
Microcompatibles Combo Package	240	215
Grafmatic	135	117
Plotmatic	135	117
Microsoft FORTRAN *w/CodeView* SALE	450	259
No Limit Library *by MEF Environmental*	129	109
Numerical Analyst *by MAGUS*	295	249
PANEL *by Roundhill Computer Systems*	295	199
RM/FORTRAN *by Ryan-McFarland*	599	399
RTC PLUS *Fortran to C by Cobalt Blue*	450	399
Scientific Subroutine Library *by Wiley*	175	135
Statistician *by Alpha Computer Service*	295	235
STATLIB.GL: *by Wiley*	295	239
STATLIB.TSF: *by Wiley*	295	239
Strings & Things *by Alpha Computer Service*	70	45

greenleaf products

	List	Ours
Greenleaf C Sampler *for QuickC & Turbo C*	95	69
Greenleaf Comm Library	185	125
Greenleaf Data Windows Library	225	155
with Source Code	395	249
Greenleaf Functions	185	125

help utilities

	List	Ours
HELP/Control *by MDS*	125	99
On-line Help *from Opt-Tech*	149	99
SoftScreen/HELP *by Dialectic Systems*	195	149

lattice products

	List	Ours
Lattice C Compiler *ver 3.2 from Lattice*	500	265
with Library Source Code	900	495
C Cross Reference Generator	50	37
with Source Code	200	139
C-Food Smorgasbord *Function Library*	150	95
with Source Code	300	179
C-Sprite *Source Level Debugger* Special Offer	175	CALL
Curses *Screen Manager*	125	85
with Source Code	250	169
dBC III	250	169
with Source Code	500	356
dBC III Plus	750	594
with Source Code	1500	1184
LMK *Make Facility*	195	138
RPG II Combo *All three items below*	1100	875
RPG II Compiler *No Royalties*	750	625
SEU *Source Entry Utility*	250	199
Sort/Merge	250	199
Screen Design Aid Utility *for RPG II*	350	309
SecretDisk II *Encryption Utility*	120	88
SideTalk *Resident Communications*	120	88
SSP/PC *Scientific Subroutine Library*	350	269
Text Management Utilities	120	88

metagraphics products

	List	Ours
LightWINDOW/C *for Datalight C*	95	79
MetaWINDOW *No Royalties*	195	159
MetaWINDOW/PLUS	275	229
TurboWINDOW/C *for Turbo C*	95	79
TurboWINDOW/Pascal *for Turbo Pascal*	95	79

micro focus products

	List	Ours
Micro Focus COBOL/2	900	729
Micro Focus Level II COBOL *w/Animator*	495	395
Level II COBOL	349	279
Level II Animator	195	155
Micro Focus Level II COBOL/ET for UNIX	CALL	CALL

CIRCLE 217 ON READER SERVICE CARD

Micro Focus PC-CICS .... *New*	1495	1189
with Micro/SPF .... *New*	1595	1269
Micro Focus Personal COBOL	149	119
Micro Focus Professional COBOL	2000	1595
Micro Focus VS COBOL/XENIX	1495	1195
Micro Focus Support Products:		
COBOL/IQ *Ad hoc Report Writer*	495	395
COBOL/IQ *for DOS 3.X Networks*	995	795
FORMS-2	295	235
SOURCEWRITER	995	795

microport products

386 Unlimited License Kit	249	209
AT Unlimited License Kit	249	209
DOSMerge286 *Run DOS and UNIX together*	149	129
DOSMerge386 *Run DOS and UNIX together*	CALL	CALL
System V/386 Combination	799	669
386 Runtime System	199	169
386 Software Development System	499	429
Text Preparation System	199	169
System V/AT Combination	549	465
AT Runtime System	199	169
AT Software Development System	249	209
Text Preparation System	199	169

microsoft products

Microsoft BASIC Compiler *for XENIX*	695	419
Microsoft BASIC Interpreter *for XENIX*	350	209
Microsoft C Compiler 5.0 *w/CodeView* ...... *SALE*	450	259
Microsoft COBOL Compiler *with COBOL Tools* . *SALE*	700	419
for XENIX .... *SALE*	995	599
Microsoft Excel .... *New*	495	295
Microsoft FORTRAN *Optimizing Compiler* ... *SALE*	450	259
Microsoft FORTRAN *for XENIX*	695	409
Microsoft Learning DOS	50	36
Microsoft Macro Assembler .... *SALE*	150	89
Microsoft Mouse *Specify Serial or Bus*	150	99
with Microsoft Windows	200	135
with EasyCAD	175	114
Microsoft Pascal Compiler .... *SALE*	300	174
for XENIX .... *SALE*	695	409
Microsoft QuickBASIC 4.0 .... *SALE*	99	59
Microsoft QuickC .... *SALE*	99	59
Microsoft Windows	99	63
Microsoft Windows Development Kit	500	299
Microsoft Word	450	269
Microsoft Works .... *New*	195	119

mks products

MKS AWK	75	65
MKS RCS *Revision Control System* .... *New*	189	155
MKS Toolkit *with MKS VI Editor*	139	109
MKS Trilogy *with AWK, CRYPT & Korn Shell* ... *New*	119	99
MKS VI *Editor by MKS*	75	65

modula-2 language

LOGITECH Modula-2 Development System	249	199
Modula-2 Compiler Pack	99	79
Modula-2 Toolkit	169	139
LOGITECH Modula-2 ROM Pkg	299	239
LOGITECH Modula-2 Window Pkg	49	39
Macro2 *Macro preprocessor by PMI*	89	79
ModBase *by PMI*	89	79
Repertoire *by PMI*	89	75
Science & Engrg Tools *by Quinn-Curtis*	75	67
Universal Graphics Library *by Quinn-Curtis*	130	119

mouse products

LOGIMOUSE BUS *with PLUS Pkg by LOGITECH*	119	98
with PLUS & PC Paintbrush	149	119
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179
with PLUS & First Publisher	CALL	CALL
LOGIMOUSE C7 *with PLUS Pkg, Specify Connector*	119	98
with PLUS & PC Paintbrush	149	119
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179
with PLUS & First Publisher	CALL	CALL
Microsoft Mouse *See Microsoft Section*		

other languages

ACTOR *by Whitewater Group*	495	419
CCS MUMPS *All varieties by MGlobal*	CALL	CALL
Marshal Pascal *by Marshal Language Systems*	189	155
Pascal-2 *by Oregon Software*	395	325
Personal REXX *by Mansfield Software*	125	99
SNOBOL4+ *by Catspaw*	95	80

other products

Carbon Copy Plus *by Meridian Technology*	195	159
Dan Bricklin's Demo Pgm *by Software Garden*	75	57
Dan Bricklin's Demo Tutorial	50	45
Fast Forward *by Mark Williams*	70	59
Instant Replay *by Nostradamus*	150	CALL
MicroTEX *Typesetting from Addison-Wesley*	295	CALL
Printer Drivers .... *New*	CALL	CALL
muMATH *by Soft Warehouse* .... *New*	300	199
Net-Tools *by BCSoft*	149	129
Norton Guides *Specify Language*	100	65
OPT-Tech Sort *by Opt-Tech Data Proc*	149	99
PC/TOOLS *by Custom Software*	49	45
Resident Expert *Specify lang by Santa Rita* .... *New*	59	55
Screen Machine *by MicroHelp*	79	59
SuperSort *by LifeStyle*	139	119

phoenix products

Pasm86 *Macro Assembler version 2.0*	195	108
Pdisk *Hard Disk & Backup Utility*	145	99
Pfantasy Pac *Phoenix Combo*	995	595
Pfinish *Execution Profiler*	395	209
Pfix86plus *Symbolic Debugger*	395	209
PforCe *Specify C Compiler*	395	209
PforC++ *Specify C Compiler and C++*	395	209
Plink86plus *Overlay Linker*	495	275
Pmaker *Make Utility*	125	78
Pmate *Macro Text Editor*	195	108
Pre-C *Lint Utility*	295	154
Ptel *Binary File Transfer Program*	195	108

polytron products

PolyBoost *Software Accelerator* ..... *Special Price*	80	54
PolyDesk III	99	72
PolyDesk III Archivist	50	42
PolyDesk III Cryptographer	50	42
PolyDesk III Talk	70	52
PolyLibrarian *Library Manager*	99	89
PolyLibrarian II *Library Manager*	149	129
PolyMake *UNIX-like Make Facility*	149	129
PolyShell .... *Special Price*	149	105
Polytron C Beautifier	50	45
PolyXREF *Complete Cross Ref Utility*	219	185
PolyXREF *One language only*	129	109
PVCS Corporate *Version Control System*	395	329
PVCS Personal	149	129

program mgmt utilities

Interactive EASYFLOW *by Haventree* .. *New Version*	150	125
PrintQ *by Software Directions*	89	84
Quilt Computing Combo *QMake & SRMS*	250	199
Sapiens MAKE	179	155
Sapiens MAKE & V8	439	379
Source Print *by Aldebaran Labs*	97	75
TLIB *Version Control System by Burton*	100	89
Tree Diagrammer *by Aldebaran Labs*	77	67

sco products

Complete XENIX System V *by SCO*	1295	994
Development System	595	499
Operating System *Specify XT or AT*	595	499
Text Processing Package	195	144
Lyrix *by SCO*	595	449
SCO Professional *1-2-3 Workalike for XENIX*	795	595
SCO XENIX-NET	595	495
XENIX System V 386 *by SCO*	CALL	CALL

softcraft products

Btrieve *ISAM Mgr with No Royalties*	245	184
Xtrieve *Query Utility*	245	184
Report Option *for Xtrieve*	145	99
Btrieve/N *for Networks*	595	454
Xtrieve/N	595	454
Report Option/N *for Xtrieve/N*	345	269
XQL .... *New*	795	CALL

text editors

Brief & dBrief Combo *from Solution Systems*	275	CALL
Brief	195	CALL
dBrief *Customizes Brief for dBASE III*	95	CALL
de *by David Livshin*	75	65
Epsilon *Emacs-like editor by Lugaru*	195	147
KEDIT *by Mansfield Software*	125	98
Micro/SPF *by PHASER SYSTEMS*	175	139
Microsoft Word	450	269
PC/VI *by Custom Software Systems*	149	99
SPF/PC *by Command Technology Corp*	CALL	CALL
Vedit Plus *by CompuView*	185	128

turbo pascal utilities

ALICE *Interpreter by Software Channels*	95	66
AZATAR DOS Toolkit *by AZATAR* .... *New*	95	85
DOS/BIOS & Mouse Tools *by Quinn-Curtis*	75	67
Flash-up *by Software Bottling* .... *New*	89	78
Flash-up Developer's Toolbox .... *New*	49	45
MACH 2 *for Turbo Pascal by Micro Help*	69	55
MetraByte D/A Tools *by Quinn-Curtis*	100	89
Science & Engrg Tools *by Quinn-Curtis*	75	67
Screen Sculptor *by Software Bottling*	125	91
Speed Screen *by Software Bottling*	35	32
System Builder *by Royal American* .... *New Version*	150	129
IMPEX *Query Utility*	100	89
Report Builder .... *New Version*	130	115
TDebugPLUS *by TurboPower Software*	60	49
Tmark *by Tangent Designs*	80	69
Turbo EXTENDER *by TurboPower Software*	85	64
Turbo OPTIMIZER *by TurboPower*	75	65
with Source Code	125	98
Turbo Professional *by Sunny Hill*	70	45
TurboHALO *from IMSI*	95	75
TurboPower Utilities *by TurboPower*	95	78
TurboRef *by Gracon Services*	50	35
TURBOsmith *Source Debugger by Visual Age*	99	89
Universal Graphics Library *by Quinn-Curtis*	130	119

wendin products

Operating System Toolbox	99	79
PCNX *Operating system*	99	79
PCVMS *Similar to VAX/VMS*	99	79
Wendin-DOS *Multitasking DOS*	99	85
Wendin-DOS Application Developer's Kit	99	85
XTC *Text Editor w/Pascal source*	99	75

xenix/unix products

Btrieve *ISAM File Mgr by SoftCraft*	595	454
C-terp *by Gimpel, Specify compiler*	498	379
c-tree *ISAM Mgr by FairCom*	395	315
dBx *with Library Source by Desktop AI*	550	419
DIRECTORY SHELL 286 *by American Mgmt Sys*	349	295
DIRECTORY SHELL 386 *by American Mgmt Sys* . *New*	495	415
DOSIX *Console Version by Data Basics*	400	349
DOSIX *User Version by Data Basics*	200	179
Lugaru *Text Editor by Epsilon* .... *New*	195	147
Micro Focus Products *See Micro Focus Section*		
Microport Products *See Microport Section*		
Microsoft Products *See Microsoft Section*		
PANEL Plus *by Roundhill Computer Systems*	795	535
REAL-TOOLS *Binary Version by PCT*	99	89
Library Source Version	599	539
Complete Source Version	999	729
RM/COBOL *by Ryan-McFarland*	1250	949
RM/FORTRAN *by Ryan-McFarland*	750	549
SCO Products *See SCO Section*		

Terms are subject to change.

CIRCLE 217 ON READER SERVICE CARD

LOWEST PRICES
Due to printing lead times, some of our current prices may differ from those shown here. Call for latest pricing.

FREE SHIPPING
Orders within the USA (including Alaska & Hawaii) are shipped FREE via UPS. Express shipping is available at the shipping carrier's standard rate with no rush fees or handling charges. To avoid delays when ordering by mail, please call first to determine the exact cost of express shipping.

CREDIT CARDS
VISA and MasterCard are accepted at no extra cost. Your card is charged when your order is shipped. Mail orders please include credit card expiration date and authorized signature.

CODs AND POs
CODs and Purchase Orders are accepted at no extra cost. No personal checks are accepted on COD orders. POs with net 30-day terms (with initial minimum order of $100) are available to qualified US accounts only.

SALES TAX
Orders outside of Ohio are not charged state sales tax. Ohio customers please add 6% Ohio tax or provide proof of tax-exemption.

INTERNATIONAL ORDERS
Shipping charges for International and Canadian orders are based on the shipping carrier's standard rate. Since rates vary between carriers, please call or write for the exact cost. International orders (except Canada), please include an additional $10 for export preparation. All payments must be made with US funds drawn on a US bank. Please include your telephone number when ordering by mail. Due to government regulations, we cannot ship to all countries.

VOLUME ORDERS
Volume orders may qualify for additional discounts. Call us for special pricing.

SOUND ADVICE
Our knowledgeable technical staff can answer technical questions, assist in comparing products and send you detailed product information tailored to your needs.

30-DAY GUARANTEE
Most of our products (excluding books) come with a 30-day documentation evaluation period or a 30-day return guarantee. Please note that some manufacturers restrict us from offering guarantees on their products. Call for more information.

MAIL ORDERS
Please include your telephone number on all mail orders. Be sure to specify computer, operating system and any applicable compiler or hardware interface(s). Send mail orders to:

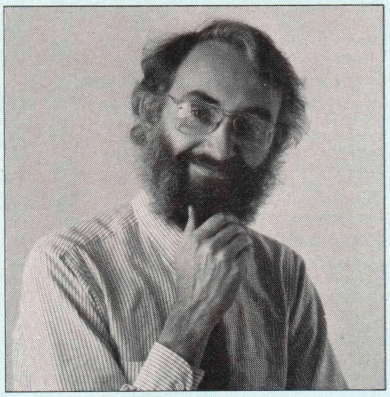
Programmer's Connection
7249 Whipple Ave. NW
North Canton, OH 44720

USA	800-336-1166
CANADA	800-225-1166
OHIO & ALASKA (Collect)	216-494-3781
TELEX	9102406879
EASYLINK	62806530
INTERNATIONAL	216-494-3781
CUSTOMER SERVICE	216-494-8899

Hours: Weekdays 8:30 AM to 8:00 PM EST.

PROGRAMMER'S CONNECTION
QUALITY INTEGRITY SUPPORT PRICE
Service

SWAINE'S FLAMES



Remember the good old days when ordinary people were afraid of computers? Then high tech became trendy. I wouldn't be surprised if Helen Gurley Brown's Cosmopolitan editorial this month began, "I just love operating systems, don't you? They're so earthy and fundamental. And 1987 was such an OS-some year!"

And Helen would be right; 1987 was OS-some. Microsoft and IBM revealed the master plan for the timed release of OS/2, the operating system for the 1990s, complete with developer's kits and seminars, and began delivering the pieces right on schedule.

Then, just when you thought that DOS was a stable development environment (OS-sified?), Microsoft and a cast of thousands announced (1) the Lotus/Intel/Microsoft Expanded Memory Specification (give a point to Quarterdeck) and (2) Windows/386 (take it away again). Now, modulo a few fatal bugs that would undoubtedly be fixed in the next release, you could stick code as well as data up above 640K, run more and bigger TSRs, switch instantaneously among existing applications; here was multitasking before OS/2.

It wasn't immediately obvious just what effect the release of Windows/386 would have on the reception of OS/2. It did seem that the peculiar benefits of OS/2 would be realized only with the arrival of OS/2-specific software, emphasizing inter-application communication or intra-application multitasking. And it depressed the Microsoft OS/2 developers, who longed to forget the 286 processor and get on with the real thing, OS/2/386, before Intel released the 486 and really depressed them.

While we were all waiting for OS/2, a few companies went ahead with 386 tools, including Phar Lap, MetaWare, Softguard, and The Software Link. Compaq offered a version of DOS that eliminated DOS's disk-file size limitations. And Digital Research delivered Concurrent DOS 386, a DOS 3.3-compatible multiuser, multitasking windowing operating system for 386 machines.

In that other universe, Apple released Multifinder, allowing users to switch smoothly between existing Mac applications. But wait; there's more.

Meanwhile, a number of developments brought Unix into prominence as an operating system for personal computers. There was the narrowing power gap between personal computers and workstations (and the entrenchment of Unix on engineering workstations). There were the steps toward standardization seen in X/Open and IEEE Posix, and the movement toward convergence of the Berkeley and AT&T versions. Although they flaunted standardization efforts, you wouldn't want to discount IBM's endorsement of Unix with its AIX (to be available on the RT, PS/2, and System 370), and Apple's ditto with A/UX. Nor would you want to discount the influence of Unix supporters Sun and NeXT, either, or all those sex, drugs, and Unix buttons at the Hackers' Conference. Then there was the growing support for the Unix graphical interface, Xwindows.

With the major operating systems all moving toward some form of multitasking and all providing some sort of windowing user interface, what the world was coming to need, many believed, were tools to ease the development process for window-oriented software, and to make it easier to port code across windowing environments. The Whitewater Group did well pitching its object-oriented language, Actor, as a tool to make Windows development easier, and began work on their Mac version.

Michael Bentley saw the same need. While reviewers gushed over Danny Goodman's book on Bill Atkinson's Hypercard (a thorough and readable book about an interesting product), Michael Bentley's *The Viewport Technician*, which promised not only to show how to develop code portable across windowing environments, but how to make that portable code efficient, got little press. Did the book deliver on its promise? Only someone who had developed code for the Amiga, GEM, Windows, and the Mac could say.

Odds and ends:

Contrary to Allen Holub's expectations, Microsoft did not deliver Quick C when promised, any more than Borland delivered Turbo C when promised.

The third edition of Daniel Remer and Stephen Elias's *Legal Care for Your Software* is now out. I'm not convinced that the authors have done their job with respect to the tricky area of ownership of reusable code in work done for hire, but generally this is a good first book in the legal issues in software development and sale. It's published by Nolo Press, 950 Parker Street, Berkeley, CA 94710.

Michael Swaine

Michael Swaine
editor-in-chief

How A C Programmer Became A Screen Star

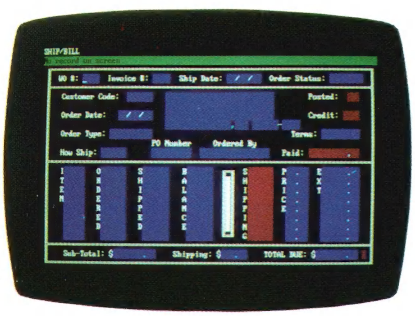
Screens, the Visible Part of Your Program.

A program is often judged by how well the screens are executed. However, the real creativity lies in what goes on behind the screens.

ScreenStar is a product that allows your real creativity to light up the screen. It reduces costly screen, window, and data validation development time.

You Take the Bows, We Write the Code.

Our natural drawing commands allow you to paint any screen imaginable. Press one key when you are satisfied and ScreenStar produces concise, commented, ready-to-compile code. This allows immediate testing of the I/O screens, including smooth, even scrolling between multiple screens.



Create or capture complex screens with data-entry filters built in.

If all ScreenStar did was turn screens into code it would be a useful tool. Yet ScreenStar also permits a wide range of field types. Some of the choices include date, alphanumeric, telephone, yes/no, dollar, time and user-definable fields.

Other valuable data-entry filters are built in, such as required field, display only, and many others. All screen fields are generated with error-checking routines.

ScreenStar Not Only Captures Your Imagination, It Captures Screens.

The memory-resident capture program converts any screen into a ScreenStar file in seconds, including those generated by programs like Dan Bricklin's Demo Program.

ScreenStar Sets the Stage for Windows.

ScreenStar comes with a complete window generating library. You design the help screens and pop-up windows. Essential ScreenStar windowing functions tie them together in one smooth package.

Curtain Call.

They may not ask for your autograph, but they will want to know how you did those screens. Screenstar is more than a screen-painting program. It is a screen processor. No professional programming environment will be complete without this product.

We know you will enjoy using ScreenStar. However, should you give it less than rave reviews, return it within 30 days for a full refund.

★ Interactive screen painting and subsequent code generation.

★ Multiple screen design and scrolling.

★ TSR screen capture program, works with any program including Dan Bricklin's Demo Program.

★ Complete window design including overlapping window functions.

★ Screens are compressed into data structures, and remain a permanent part of the program. No messy data files to look for.

Price - $99

W/Source add $99

Audition Our Product Today. Call:

CIRCLE 218 ON READER SERVICE CARD